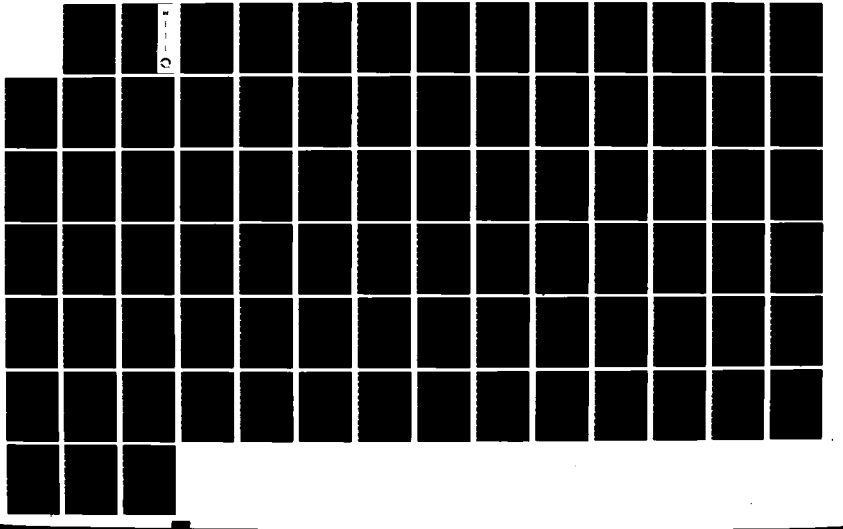
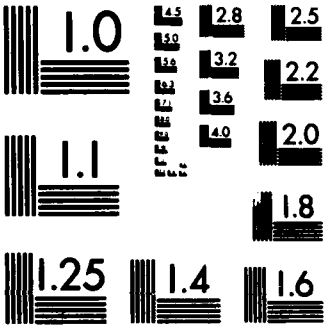
AD-A134 999 APPLICATION OF HIERARCHICAL DATA STRUCTURES TO GEOGRAPHICAL INFORMATION SYSTEMS(U) MARYLAND UNIV COLLEGE PARK COMPUTER VISION LAB H SAMET ET AL. 1/1

UNCLASSIFIED 30 SEP 83 TR-1327 ETL-0337 DAAK70-81-C-0059 F/G 8/2 NL





ETL-0337

Application of hierarchical data structures to geographical information systems (Phase II)

Hanan Samet

Azriel Rosenfeld

Computer Vision Laboratory
University of Maryland
College Park, MD 20742

SEPTEMBER 1983

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

Prepared for
U.S. ARMY CORPS OF ENGINEERS
ENGINEER TOPOGRAPHIC LABORATORIES
FORT BELVOIR, VIRGINIA 22060

83

E T L

U.S. ARMY ENGINEER TOPOGRAPHIC LABORATORIES ETL

Destroy this report when no longer needed.
Do not return it to the originator.

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

The citation in this report of trade names of commercially available products does not constitute official endorsement or approval of the use of such products.

Final Report on
Contract DAAK70-81-C-0059/P00007

APPLICATION OF HIERARCHICAL DATA STRUCTURES TO GEOGRAPHICAL INFORMATION SYSTEMS PHASE II

Submitted to:
U.S. Army Engineer
Topographic Laboratories
Fort Belvoir, VA 22060
Attention: Mr. Joseph A. Rastatter

Submitted by:
Computer Vision Laboratory
Center for Automation Research
University of Maryland
College Park, MD 20742

Principal Investigators:
Hanan Samet
Azriel Rosenfeld

September 30, 1983

PREFACE

This report documents the research conducted under Phase II of Contract DAAK70-81-C-0059/P00007. The report was prepared for the U.S. Army Engineer Topographic Laboratories (ETL), Ft. Belvoir, Virginia 22060. The Contracting Officer's Representative was Mr. Joseph A. Rastatter.

This report was prepared by Hanan Samet, Azriel Rosenfeld, Clifford A. Shaffer, and Robert E. Webber.

SUMMARY

This document is the final report for Phase II of an investigation of the application of hierarchical data structures to geographical information systems, conducted under Department of the Army Contract DAAK70-81-C-0059/P00007. The purposes of this investigation were twofold: (1) to construct a geographic information system based on the quadtree hierarchical data structure, and (2) to gather statistics to allow the evaluation of the usefulness of this approach to geographic information system organization.

To accomplish the above objectives, in Phase I of the project a database was built that contained three maps supplied under the terms of the contract. These maps described the flood plain, elevation contours, and landuse classes of a region in California. The map regions were represented in quadtree form, and algorithms were developed for basic operations on quadtree-represented regions (set-theoretic operations, point-in-region determination, region property computation, and submap generation). The efficiency of these algorithms was studied theoretically and experimentally.

On Phase II of the project, the following additional tasks were performed:

(a) Query Language. Design of a high-level query language permitting easy interaction with the database by users, thus making the quadtree representation transparent to the users.

(b) Database updating. Development of algorithms for addition, deletion, and editing of data items in a quadtree-encoded database.

(c) Point and linear feature data. Quadtree-like representations of point and linear feature data, extracted from the same geographic region, were also constructed. Algorithms were developed for interfacing between these representations and the quadtree-represented areas.

TABLE OF CONTENTS

page

1. Introduction 1

2. The database query language 2
   2.1 An overview of the query language 2
   2.2 The query language syntax 3
   2.3 A demonstration of the query language 9
   2.4 On the timing of the query language demonstration . 30

3. The quadtree editor - a tool for database update 35
   3.1 An overview of the quadtree editor 35
   3.2 Quadtree editor commands 37
   3.3 Implementation of the quadtree editor commands 44
   3.4 A demonstration of database updating 47

4. The quadtree memory management system 53
   4.1 The user's view of the memory management system ... 53
   4.2 Implementation of the memory management system 56

5. Point and line data 58

6. Conclusions and plans 61

Appendix: Facilities used 62

Bibliography on quadtrees 63

FIGURES

page

1. The geological survey map of the Russian River valley . 14
2. The floodplain map 15
3. The landuse map 16
4. The topography map 17
5. The map named 'zz' 18
6. The map named 'center' 19
7. The map named 'low' 20
8. The map named 'step1' 21
9. The map named 'final' 22
10. The road map 23
11. The city border map 24
12. The powerline map 25
13. The railroad map 26
14. The map named 'lowroad' 27
15. The house map 28
16. Intersection of the house map with 'low' 29
17. The floodplain map with marked revisions 50
18. The landuse map 51
19. The updated landuse map 52

TABLES

page

1. Timings for example demonstration 32
2. Sizes of maps referred to by Table 1 33
3. Timings for intersection task 34

1. Introduction

This project is concerned with the applicability of a class of hierarchical data structures, known as "quadtrees", to the representation of cartographic data. Section 2 details the database query language that defines how the system appears to the user. Section 3 presents the quadtree editor which allows the user to directly edit and update a map. Section 4 describes the quadtree memory management system which allows the user to manipulate maps that cannot fit in core. Section 5 discusses the usage of the quadtree memory management system to store maps of point data (e.g., a map of house locations) as well as line data (e.g., a map of road layouts). Section 6 presents our conclusions and future plans. At the end of the report is a bibliography on quadtrees.

The facilities used on the project are described in the Appendix. In particular, the display deviced used by this project is a Grinnell GMR-27 Display Processor (see [Kirb79] for information pertaining to use of this device at the University of Maryland), referred to in the rest of this report as the Grinnell.

2. The database query language

2.1. An overview of the query language

The query language is an English-like keyword-based interface between the database user and the database system. The query language allows the user to display, window, and construct submaps from individual maps, perform set-theoretic operations on groups of maps, and extract various kinds of information from a map (e.g., find the total area of a map, list the polygons in a map, etc.). It also allows the user to access the quadtree editor (described in Section 3) which enables the user to update maps and draw new ones.

The query language is embedded in the University of Maryland version of FRANZ LISP ([Fode80],[Alle82]). The entire database system can be viewed operationally as a query language that is interpreted by LISP as LISP functions which call C functions (i.e., functions coded in the programming language C) to actually process the maps. Thus all of the algorithms of the database are coded in C, and LISP merely serves as a convenient front end for translating the query language into calls to C functions. Although normally it is necessary to enclose a function call in parentheses when using LISP, the particular LISP we are using interprets an input line containing three or more words as being implicitly enclosed by parentheses. We make use of this device to give the interface a more natural appearance to users who are not used to LISP.

The query language is keyword based, which means that the database ignores words that it doesn't understand. This has the advantage that one can insert words and phrases (e.g., articles like "the" and "an") to give the command a more natural appearance, or one can ignore unnecessary phrases and just type the minimum to cause the appropriate commands to be executed. This added flexibility is bought at the cost of more obscure error messages resulting from the misspelling of a keyword. In order to allow the user to customize his interface with the database, there are commands that allow keywords to be changed.

The syntax of the database interface is explained in Section 2.2. In Section 2.3, an example of an interactive session with the database is presented. Section 2.4 contains timing statistics gathered when using the database system.

2.2. The query language syntax

A description of the query language syntax is presented. Curly braces { } are used to indicate phrases in the command where a sequence of non-keywords can be inserted. These words are used to add clarity to the query. Some standard sequences are shown in the given command forms.

Words enclosed in angle brackets <> are syntactic units. A syntactic unit is simply something which, when the query is typed, is replaced by its definition. For example, suppose we had a syntactic unit <color> whose definition was red, green, or blue. Then whenever there was a query whose syntax contained the symbol <color> we would actually type our choice of red, green, or blue. The query language is presented by describing each syntactic unit.

2.2.1. <command>

The portion of the query language that corresponds to an English sentence is a <command>. All other syntactic units correspond to words or phrases. The following are the allowable forms for a command:

```
Please { explain } <syntactic_unit> { }
Use { the Grinnell at } <window> { }
Measure { points from the lower left corner of } map { }
Measure { points from the } global { origin }
Enter <file_name> { into database }
Display <map> { on Grinnell }
Display <map> { on Grinnell starting from } <point> { }
Display { the } value { of } <number> { }
Let <name> { } denote { } <object> { }
Let <name> { } rename { } <map> { }
Describe { the type of this } <name> { }
Forget { about the meaning of this } <key_word> { }
List { all the } classes { on } <map> { }
List { all the } polygons { on } <map> { }
Edit { } <map> { with the database editor }
Move { to } <point> { }
```

Perhaps most useful to the beginning user is the Please command. The request

Please explain <command>

gives a list of the allowable forms for a <command>. In general, one uses the Please command as a help function which gives information about syntactic units.

The following three commands are used to initialize the database system: the Use command, the Measure command, and the Enter command. The Use command enables the user to specify a portion of the display device (in our case, a Grinnell GMR-27) that is to be used in displaying the map.

The Measure commands tell the database system how the user wishes coordinates of points to be interpreted. One can either measure locations from the lower left-hand corner of the map or from some external global origin (defined when the map was built or last edited). The Enter command places the name of a quadtree file into the database's list of known quadtree files. The Enter command also checks the file to make sure that it is a quadtree file.

There are three different types of Display commands. The first type simply causes a map to be displayed in the portion of the display device that was defined by the Use command. This command places the lower left-hand corner of the map in the lower left-hand corner of the display region. The second type of Display command allows the user to specify some other part of the map to be placed in the lower left-hand portion of the display region. This is particularly useful for displaying maps that are larger than the display region. The third type of Display command allows one to display the value of a clause (expression) without first having to bind the value to a name.

There are four commands that allow the user to manipulate the names that the query language interpreter knows about (these are in addition to the Enter command described above): two forms of the Let command, the Describe command, and the Forget command. The first form of the Let command enables the user to associate a name with an object. The value of the name becomes the value of the object at the time the Let was performed. The second form of the Let command permits the user to name a map and at the same time remove the old name of the map. This is particularly useful because each map name corresponds to a file in the operating system and one doesn't want to clutter up one's directories with the names of many temporary maps. The Describe command tells the user what the query language interpreter knows about a particular name (generally its type and something about its value). When the Describe command is applied to a quadtree file, the quadtree file's header information is displayed on the terminal. The Forget command causes the interpreter to forget a previous meaning that had been associated with a name.

Finally there are four miscellaneous commands: two types of List commands, the Edit command, and the Move command. The first type of List command allows the user to get a list of the classes that are used by a particular map. Each class corresponds to a color on the map. The second type of List command provides the user with a list of polygons (simply-connected regions) in a map. Note that a polygon name is a class name concatenated with the x and y coordinates of some point inside the polygon. The Edit command allows the user to access the the quadtree editor (QED)

described in Section 3. The Move command causes a cursor on the display device to be moved to a specified location.

2.2.2. <syntactic unit>

The possible <syntactic_unit>s constitute the allowable parameters to the Please command. They also correspond to the actual syntactic units of the query language grammar augmented by two additional values, <system> and <syntax>, whose explanations tell how to interpret the results of a Please command. The following are the allowable <syntactic_unit>s:

<system>
<syntax>
<class>
<command>
<cplist>
<file_name>
<key_word>
<map>
<name>
<number>
<object>
<point>
<polygon>
<syntactic_unit>
<window>

The explanation of a <syntactic_unit> is an abbreviated version of its description in this document. Note that <syntactic_unit>s are meant to be used with angle brackets enclosing them. Thus for an abbreviated version of Section 2.2.1, one would type

Please explain <command>

An error would result from a request to explain anything other than one of the <syntactic_unit>s that are listed above. Failure to use angle brackets also generates an error. Clearly a more forgiving version could be provided if ever needed.

2.2.3. <number>

A number can be represented in decimal format or by a <name> that denotes a number. It can also have one of the following forms:

{ the } area { of } <map>
{ the } perimeter { of } <map>

The area form allows one to calculate the total area of the non-white parts of a line, point, or area map. The perimeter form is used to calculate the perimeter of the non-white

parts of an area map.

2.2.4. <name>

A name can be represented by a letter followed by any sequence of characters. Names are used to denote keywords in the query language and the values of objects created by the user. A name can be used wherever the object that the name denotes can appear.

2.2.5. <point>

A point represents a location on the map. A map location can be specified in one of the following ways:

{ the point where x = } <number> { and y = } <number>
{ the point at the } cursor

The specification of a point by the value of its x and y coordinates is useful when these values are known. Often it is difficult to determine accurate x and y values by just looking at the screen. When an accurate point value is needed, the most convenient way to get it is to position the cursor at the location on the map that is desired. The location of the cursor can then be referred to using the keyword "cursor".

2.2.6. <window>

A window describes a rectangular region. A window can be specified in one of the following ways:

{ from } <point> { extending } <number> { by } <number>
{ the smallest } window { for } <map>

In the first specification, <point> refers to the coordinate of the lower left corner of the window. It is followed by the width and the height of the window. The smallest window for a map is the smallest rectangle that encloses all the non-white regions in the map.

2.2.7. <file name>

<file-name> is a special type of name which is interpreted as the name of a file in the operating system. Only such names are valid parameters to the Enter command.

2.2.8. <map>

A map can be denoted by a <file_name> that has been entered into the database. It can also be constructed by one of the following phrases:

{ the } intersection { of } <map> { with } <map>
{ the } union { of } <map> { with } <map>
{ the } windowing { of } <map> { with } <window>
{ the } map { formed from } <cplist> { in } <map>
{ the } points { in } <number> { of } <point> { in } <map>

The first two phrases construct the obvious set-theoretic results. There is no phrase to construct the set-theoretic complement of a map, because this can be done easily in the quadtree editor described in Section 3. The windowing construction creates a map whose lower left-hand corner is the lower left-hand corner of the window and which has had all the area that lies outside the window painted white. A <cplist> is a collection of classes and polygons. The map formed from a <cplist> looks exactly like the original map except that all regions not specified in the <cplist> have been painted white. The <cplist> construct in conjunction with union, intersection, and windowing will allow the user to create maps containing any collection of polygons from any set of maps he desires. The points construction is used to build a map of the points within a distance <number> of a location <point> in a point map <map>.

2.2.9. <object>

The types of objects that can be associated with a name are the following:

<class>
<key_word>
<map>
<number>
<point>
<polygon>
<window>

2.2.10. <class>

A class is used to refer to a collection of regions in a map that have the same color. There are two kinds of objects that have colors associated with them. One is the polygon, which is a simply-connected region of one color, and the other is the point, which is a location that can have a color. Thus classes can be specified in the following two ways:

{ the } class { of } <polygon>
{ the } class { at } <point> { on } <map>

The word 'class' comes from the concept of a landuse class - for example, all of the wheatfields might form a class. Here that notion has been extended so that each topography

level has a predefined class - e.g. all the polygons with elevation below 100 feet are part of the class level1. Additionally there is the class of polygons which are white (no information or binary value '0'). All classes have a unique color value, and new classes can be created on the fly (and renamed if desired) by giving a polygon an unused color.

2.2.11. <polygon>

A polygon is a simply-connected region in a particular map. Any point in a polygon can be used to denote a polygon. Thus the internal form for a polygon is a point value concatenated with a map name. Sometimes it is useful to have a unique name for a polygon. A typical case might be when polygons of the same class are derived from two different methods, and the user wishes to know if they are the same polygon. In those situations, an arbitrary ordering is imposed on the points inside the polygon (corresponding to the order in which a tree traversal would find them) and the least point (according to this ordering) that is inside the polygon is used to uniquely denote the polygon (when concatenated with the polygon's map name). Quite often, however, a unique name is not needed. Since calculating unique polygon names is expensive, the following two forms are given to allow the user to specify a polygon and whether or not the name should be unique:

{ the } polygon { at } <point> { on } <map>
{ the } unique { } polygon { at } <point> { on } <map>

2.2.12. <key word>

The keywords recognized by the query language are any words that appear outside the curly braces in a syntax description. These include the syntactic units (see Section 2.4) and the following: area, class, classes, cursor, denote, global intersection, map, perimeter, polygon, polygons, rename, union, unique, value, window, windowing. Also included are: Describe, Display, Edit, Enter, Forget, Let, List, Measure, Move, Please, Use.

2.2.13. <cplist>

A cplist is a nonempty list of classes and polygons. This allows the user to specify any arbitrary subset of polygons from an area map. This is a portion of the grammar that could easily be extended to include various operators for building a list of classes and polygons using set-theoretic operations in conjunction with the total list of classes or polygons in a map. However, so far, no need has been found for such a capability.

2.3. A demonstration of the query language

Below is an example of an interactive session between a user and the database. Our database is a collection of maps relating to the Geological Survey Map shown in Figure 1. Phase I of our project used three maps that represented three different partitionings of Figure 1, and are shown in Figures 2, 3, and 4. Note that these figures represent multi-color maps. The first portion of this demonstration will be concerned with showing how the tasks of Phase I can be done using the query language. We assume that the database has already been invoked via the appropriate system calls.

```
help
```

The one word phrase help is used to remind the user of how to use the Please command. This ability is not actually part of the formal query language, but exists to maintain compatibility with the user's expectation that if he types help at a system, he will be told something useful.

```
Please explain this <system>
```

This is the official way to find out how to get a list of the system commands.

```
Please explain <command>
```

And this is the official way to get the list of the system commands. These are the same commands listed in Section 2.2.

```
Enter flood into the database
Enter land in
```

These two Enter commands verified that flood and land are names of files that contain maps. A connection between these names, flood and land, and those files, has been made by the query language interpreter. Note that flood is the name of the floodplain map and land is the name of the land-use map.

```
Let be denote denote
```

This allows us to use "be" anywhere we could use the keyword "denote". This illustrates how the user can tailor the query language to his own taste.

```
Let extra be land
```

Now both extra and land can be used as names for the landuse map. Note that this does not create a new map.

Describe extra please

The Describe command allows the user to verify that the previous Let command actually worked. Perhaps a more realistic usage of the Describe command would be to ask for a description of a name that had been set thirty commands ago and whose exact usage had now been forgotten.

Let x be 100
Let y be 400

We can now use x and y to denote 100 and 400 respectively.

Let z be the polygon at x y in extra

Thus the above command causes z to refer to the polygon at 100 400 in the landuse map.

Use 0 0 512 512 to open Grinnell

This tells the database to use a window size of 512 by 512 (the entire Grinnell screen) for displaying a map. This is the standard size for our maps.

Let zmap rename the map for z from extra

Now zmap is the name of a map that is white except for a polygon at 100 400 that has been copied from the landuse map.

Let center be map of class at 100 400 in flood from flood

Now center refers to a map that contains only the polygons of the floodplain map that have the same color as that of the polygon at 100 400 (in the floodplain map).

Display center please

This enables the user to see the new map called center. Seeing the map we named center, we note that it actually contains the left bank of the floodplain.

Let left rename center

Thus, we rename it left.

Display center please

This results in an error message because the rename form of the Let command does an implicit Forget command on the file name that corresponds to its object.

Display left please
Let frame be the smallest window enclosing zmap

Since zmap is a map containing only the polygon z, frame now denotes the smallest rectangle that encloses the polygon z.

```
Enter top into the database
```

Note that top is the name of the topography map.

```
Let zz be the windowing of land with frame
```

We have now created a new map that is smaller than land and includes that portion of the landuse map that was within the smallest enclosing rectangle of the polygon z. Thus zz constitutes a map of z with some surrounding context (as opposed to a map of z surrounded by white).

```
Display zz quickly!
```

Note that the word "quickly" has no special meaning to the system, although in this case the display is indeed done quickly. A picture of zz is shown in Figure 5.

```
Use 256 256 128 128 to open Grinnell
```

Here we see one of the benefits of being able to specify that output is done in a particular part of the display region. After execution of the next command, we will be able to have two copies of the map zz on the screen. Using this type of facility, one could edit a map while simultaneously viewing a copy of the original map. Unfortunately, the current implementation only allows manipulation of the map in the last window created using the Use command. A more sophisticated system might allow one to edit more than one map at the same time, much as some text editors allow the editing of more than one textfile during the same invocation.

```
Display zz please
Use 0 0 512 512 to open Grinnell
```

Now we resume using the entire Grinnell screen.

```
Display the value of the area of left
```

Recall that left is the name of the left bank in the floodplain.

```
Display the value of the perimeter of left
```

These Display commands are printing values to the terminal.

```
Let center be a map of class at 100 300 on flood from flood
Display center please
```

This time we make center from the correct portion of the floodplain (as shown in Figure 6).

```
Let low be the map of level1 on top
Display low please
```

Low is a convenient name for a map of the lowest contour level in the topography map. Note that level1 is a class name that is currently hardwired into the system and indicates the first level in a topography map (as shown in Figure 7).

```
Let step1 be the intersection of land with low
Display step1 please
```

Now we are creating a new map (Figure 8) that contains those portions of the landuse map that lie inside the boundaries of level1 in the topography map.

```
Let final be the intersection of step1 with center
```

The result of the previous operation is in turn intersected with the center region of the floodplain map.

```
Display final please
```

Now the results are on the Grinnell screen (and Figure 9).

This particular result was one of the computations performed by the system during Phase I and reported in [Rose82a]. Recalling the sequence of operating system commands that were necessary to perform the same calculation before the query language was implemented, one can see the merits of having such a query language associated with a database system. Such an interface makes the commands more easily understood and provides a way of using the system that is transportable across different operating systems.

Phase II of our project required integration of line and point data into the database. From the map of Figure 1, four maps of line data were extracted. The most interesting one is shown in Figure 10, and consists of the roads shown on Figure 1. Figures 11, 12, and 13 show other linear features on Figure 1. The Power Lines Map and the Railroad Map are conceptually similar to the Road Map (although much simpler). The City Border Map contains a closed curve (the line begins and ends on the same point). This information could have been stored as an area map with the region within the city border as a single polygon (a 'black' area) and the outside region another polygon (a 'white' area). Converting between line maps and area maps in such a way is a task outside the scope of Phase II of this project; however this might be a suitable task for future work. Line maps are en-

tered into the database and displayed by the same commands as are area maps, as is shown in the following commands.

```
Enter road into the database
Display road please
```

Line maps can be intersected with area maps, but not with point maps or other line maps.

```
Let lowroad denote the intersection of road and low
Display lowroad
```

The result of this intersection is shown in Figure 14.

```
Display the value of the area of lowroad
```

The above returns the number of pixels in the digitization of the lines in the map lowroad. This yields a rough estimate of the length of the lines in the map.

We also have a set of point data, shown in Figure 15, which corresponds to the location of the houses on the map in Figure 1. This map can also be entered and displayed in a natural manner.

```
Enter point into database
Display point please
```

It is possible to display map expressions, as well as map names, as demonstrated by the following command.

```
Display the intersection of point with low
```

The result of this command is shown in Figure 16. As with line maps, point maps can only be intersected with area maps. It is also possible to ask for the area of a point map, which has the natural interpretation of returning the number of points in the map.

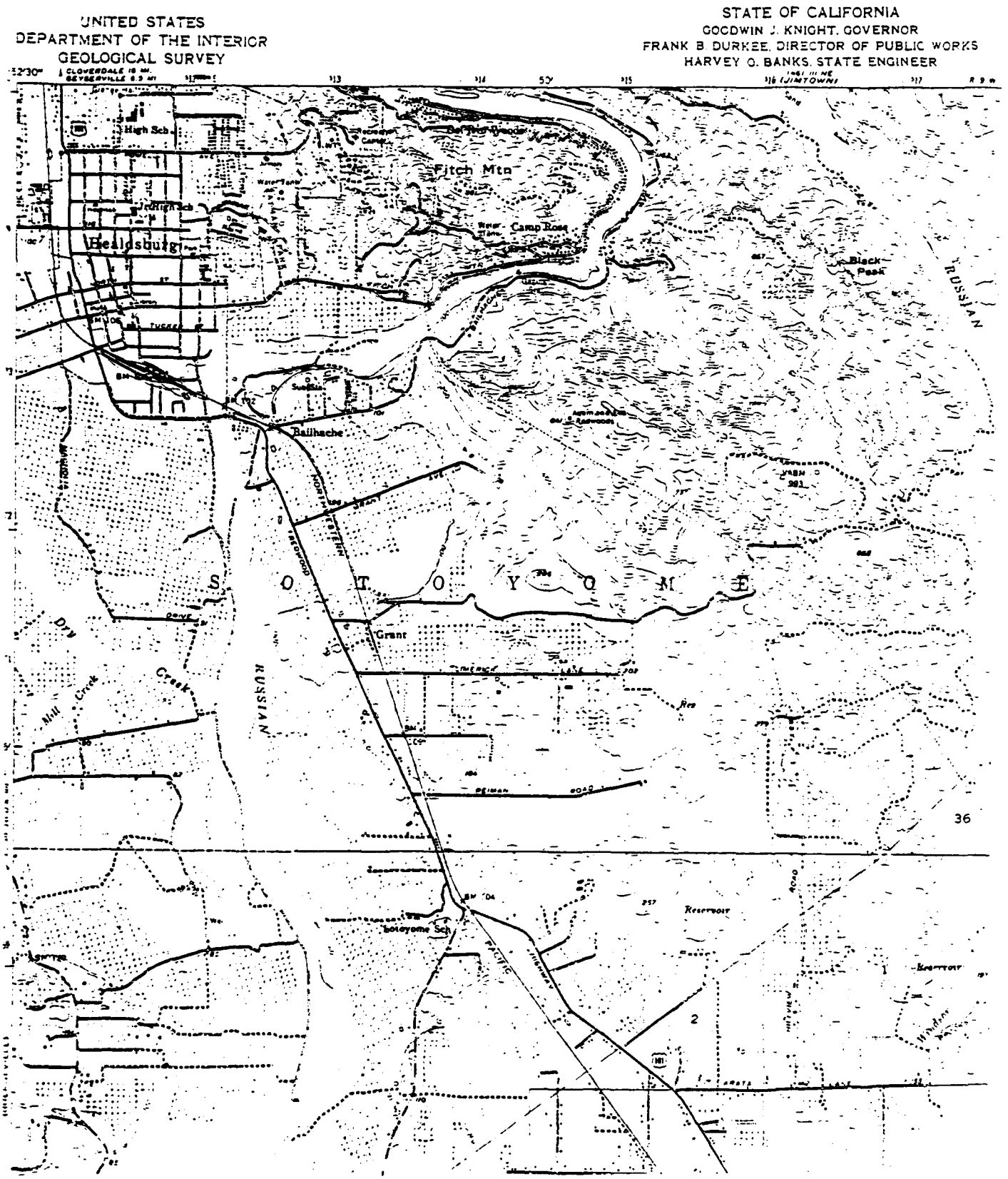


Figure 1. The geological survey map of the Russian River valley.

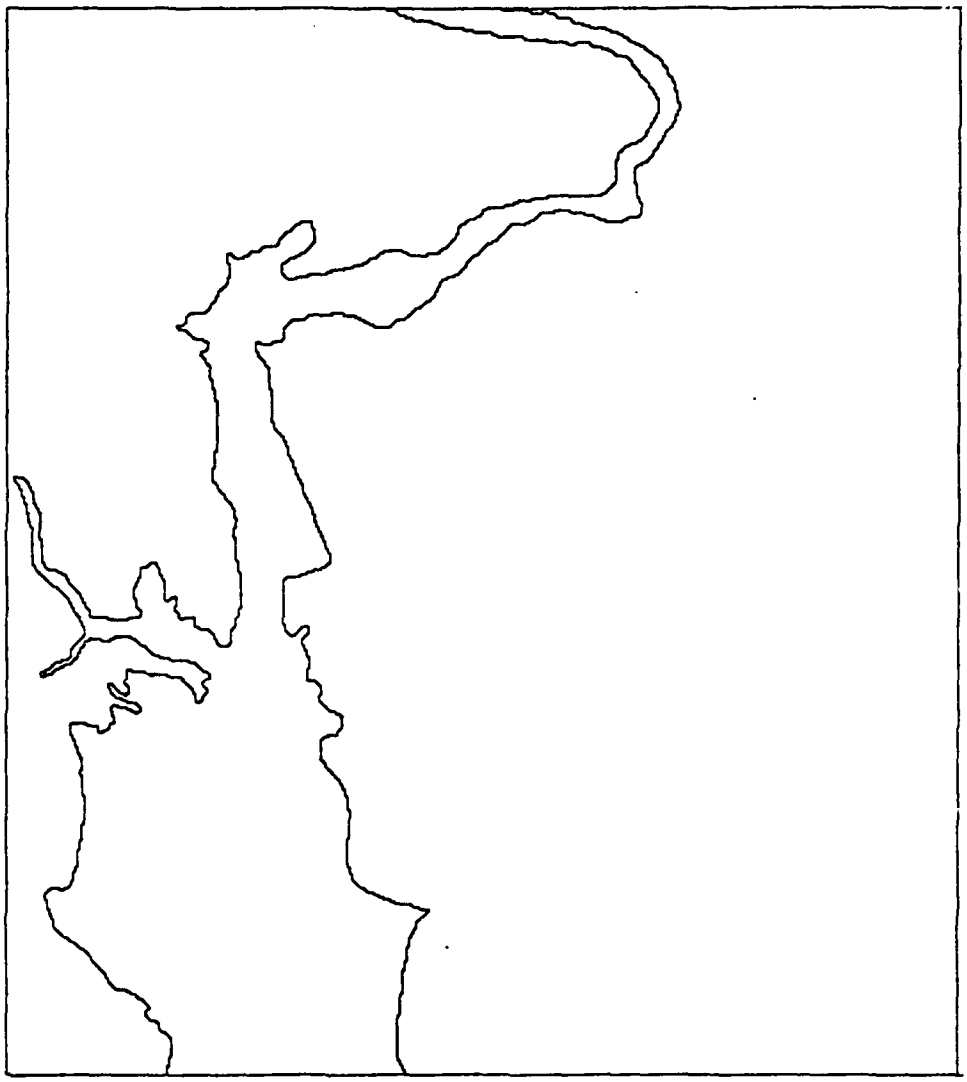


Figure 2. The floodplain map.

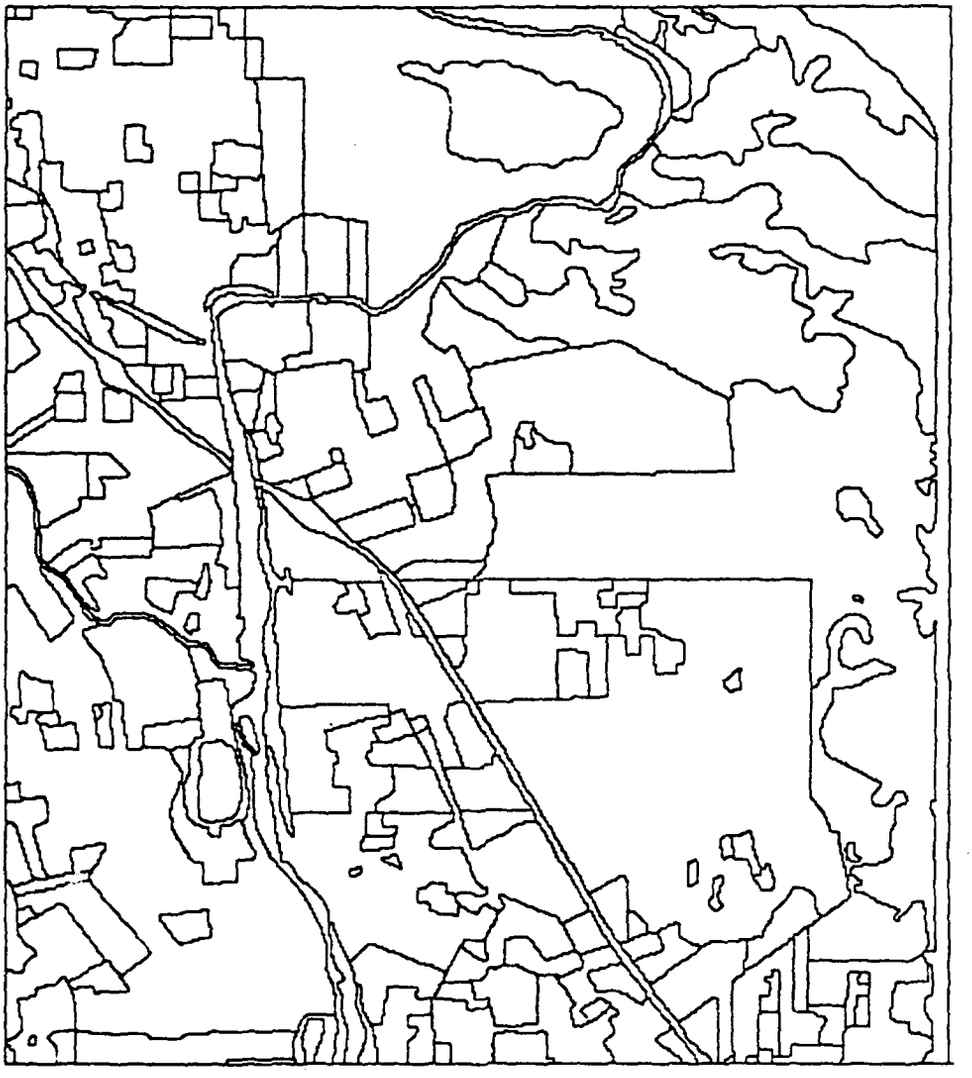


Figure 3. The landuse map.

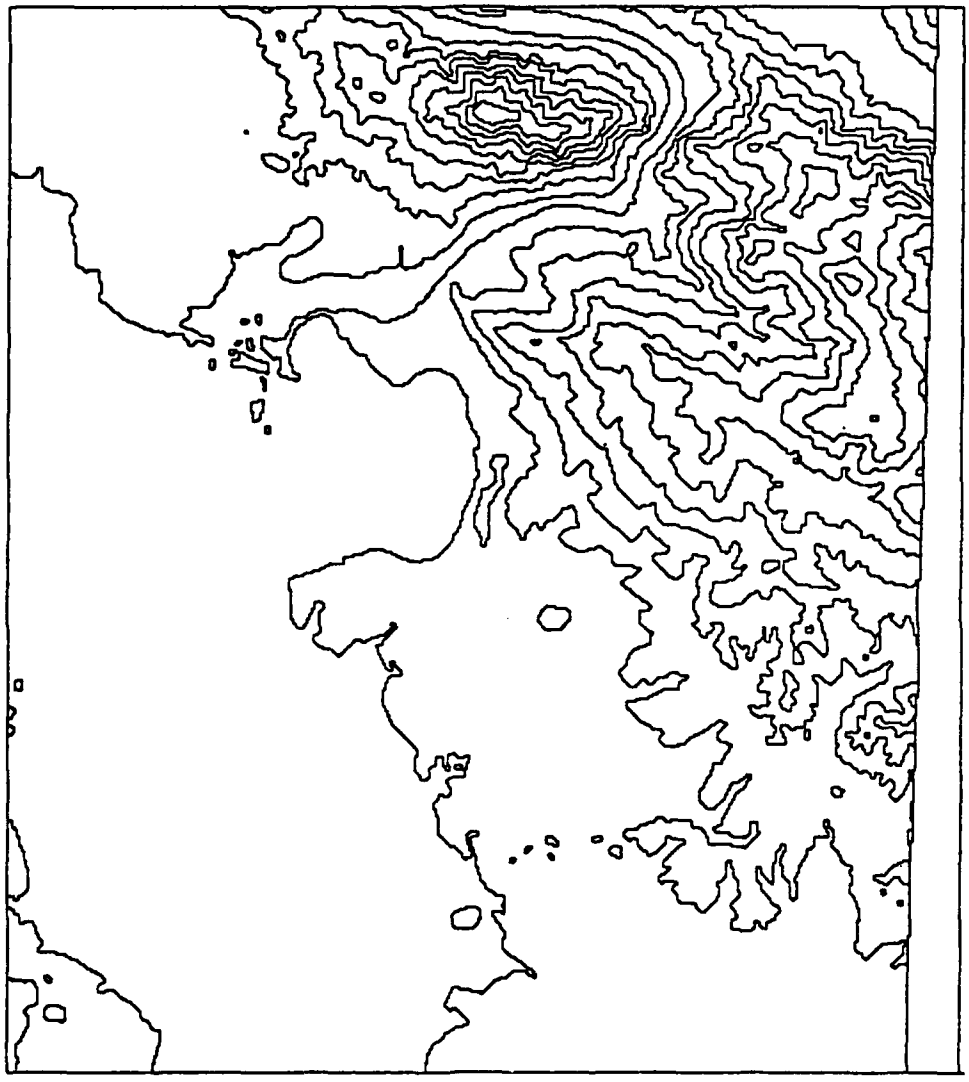


Figure 4. The topography map.

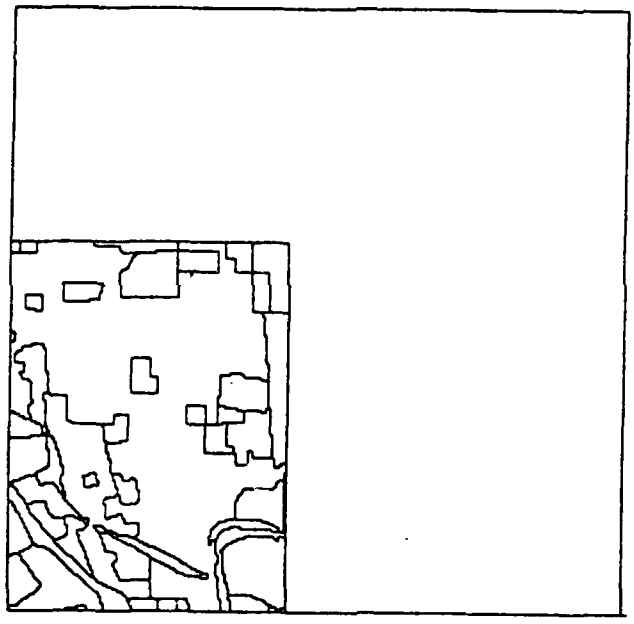


Figure 5. The map named 'zz' (a window of the landuse map).

Figure 6. The map named 'center' (the center region of the floodplain map).

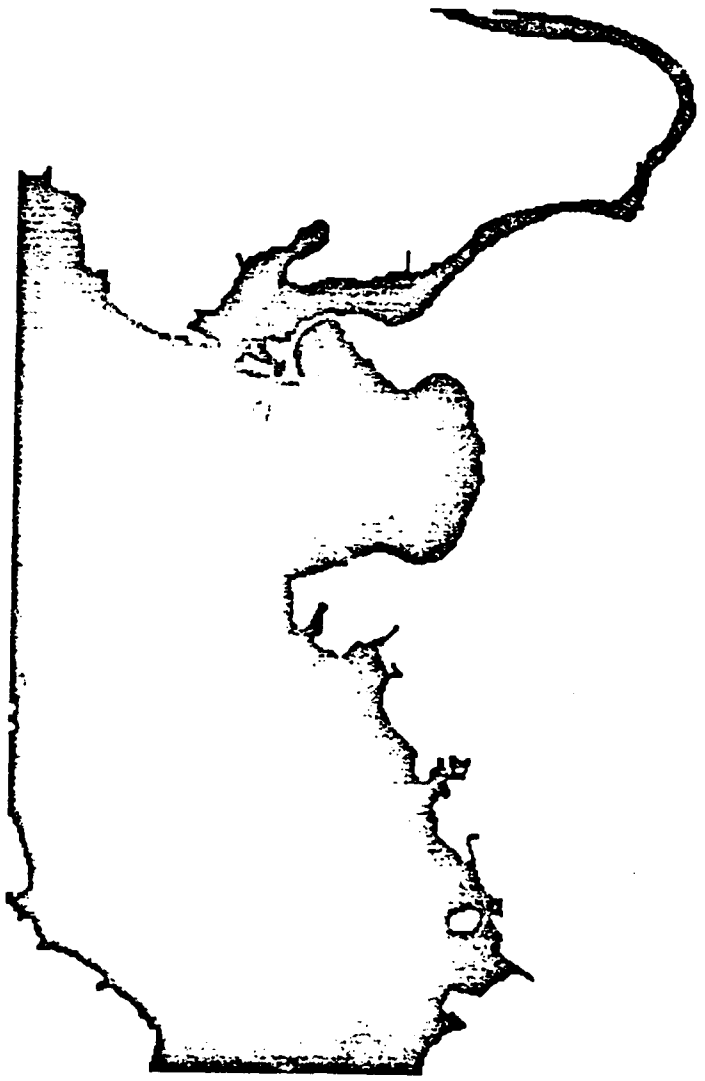


Figure 7. The map named 'low' (the lowest contour of the topography map).

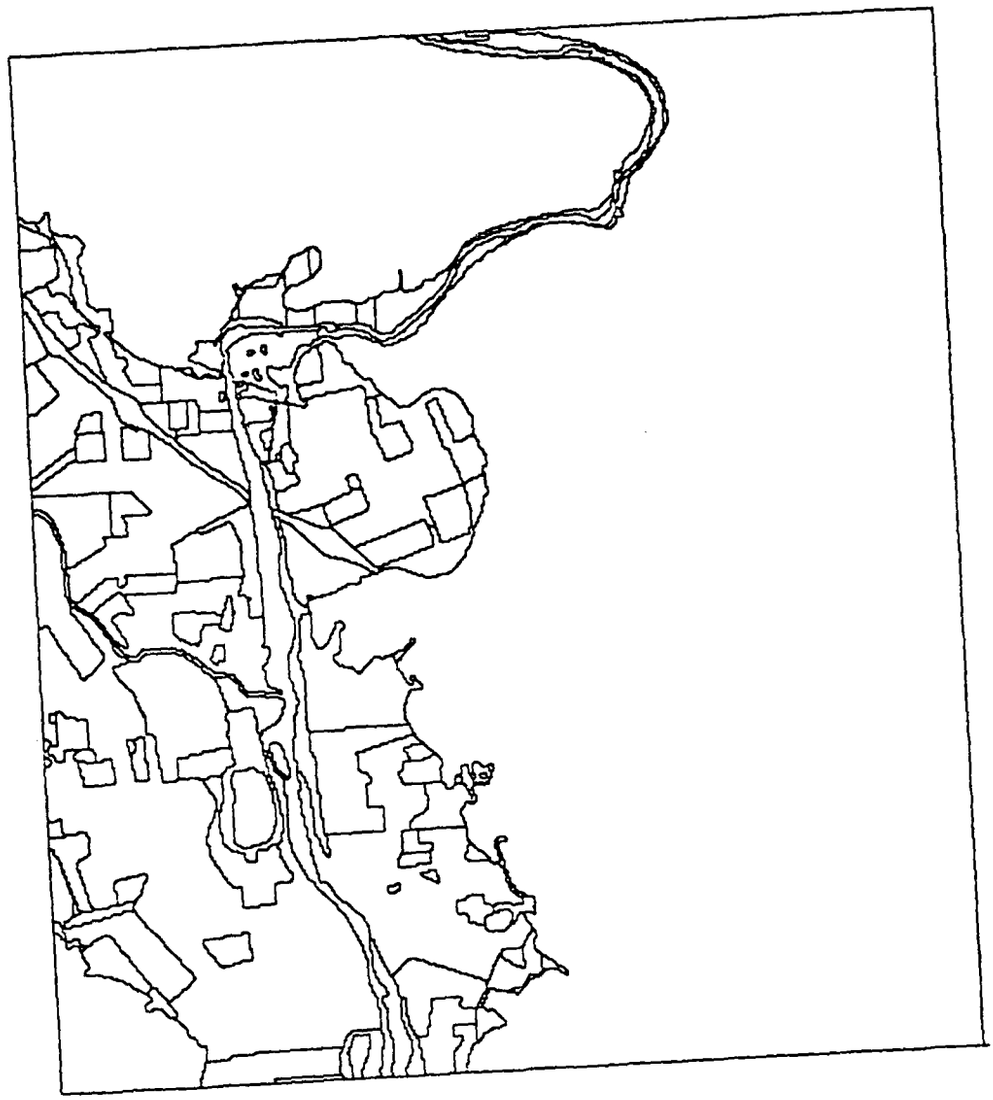


Figure 8. The map named 'step1' (the intersection of the landuse map with the map named 'low').

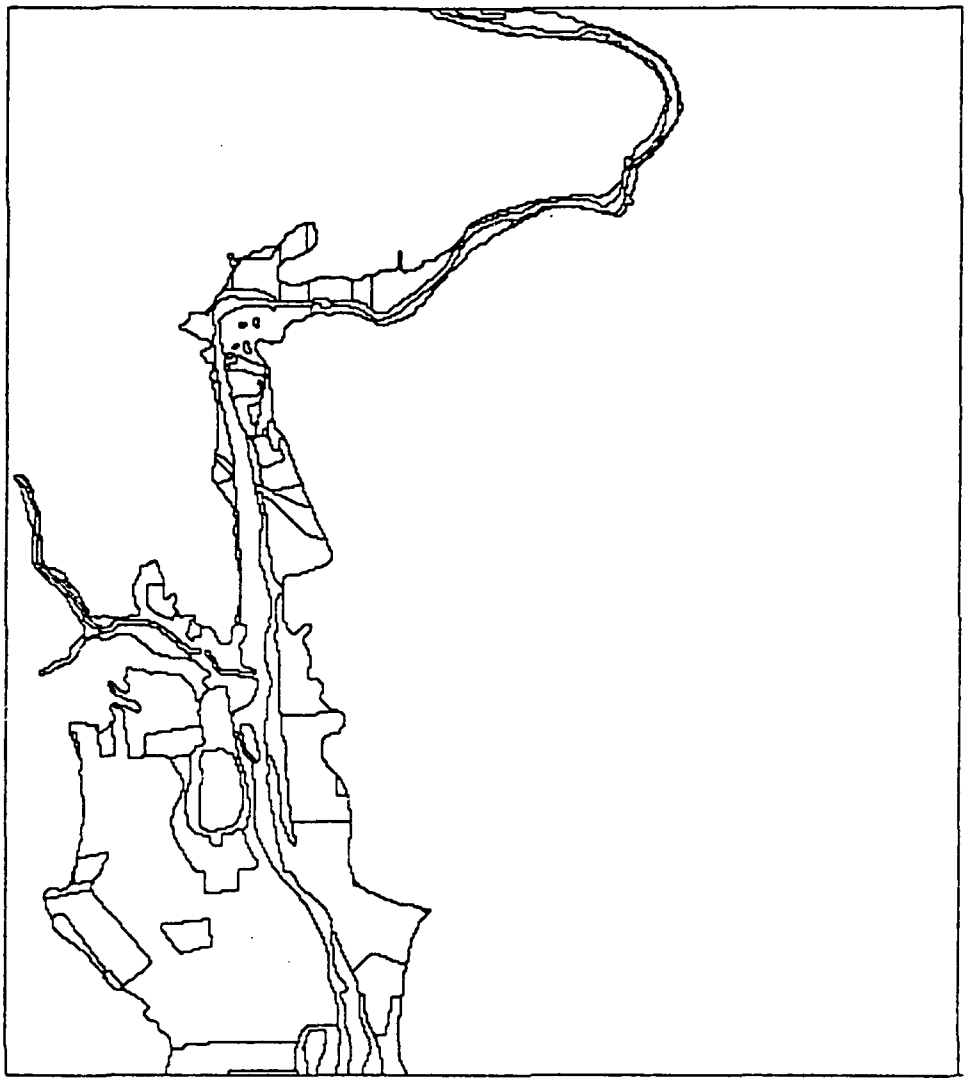


Figure 9. The map named 'final' (the intersection of 'step1' with 'center').

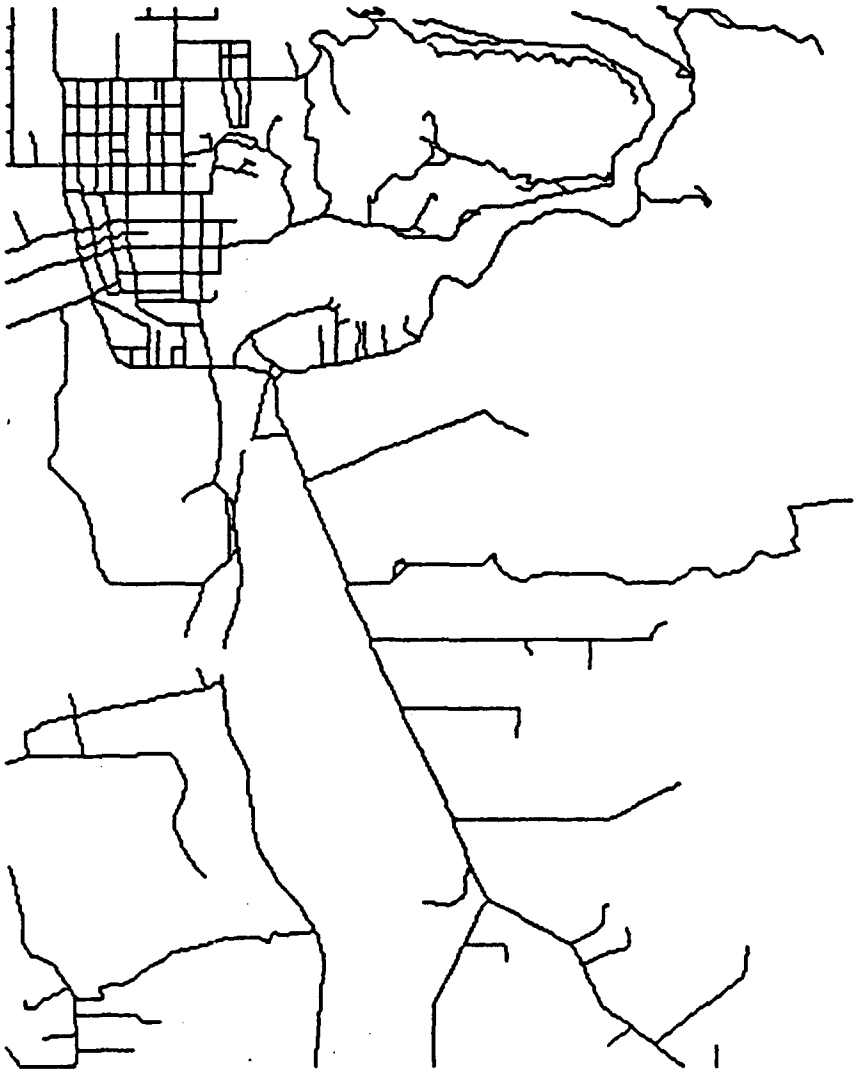


Figure 10. The road map.

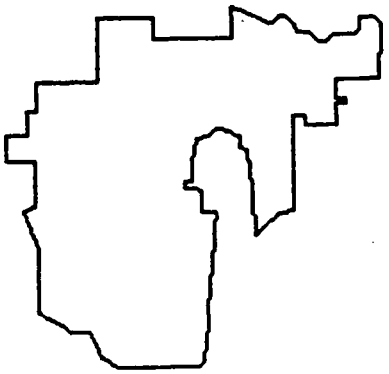


Figure 11. The city border map.

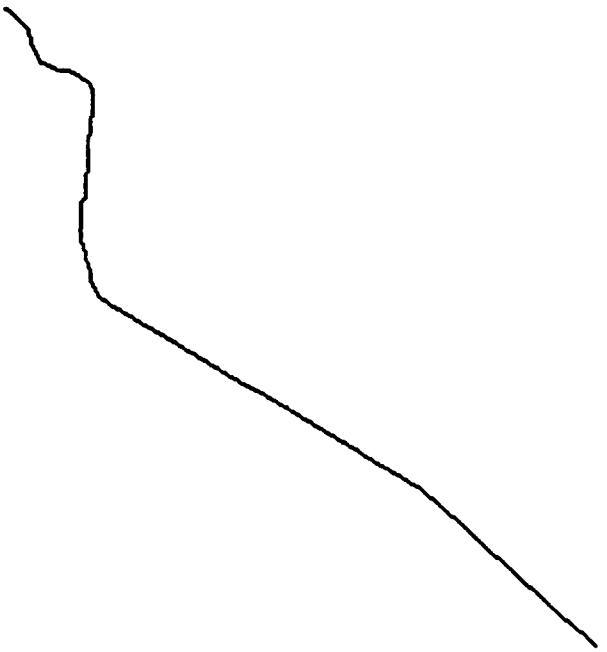


Figure 12. The powerline map.

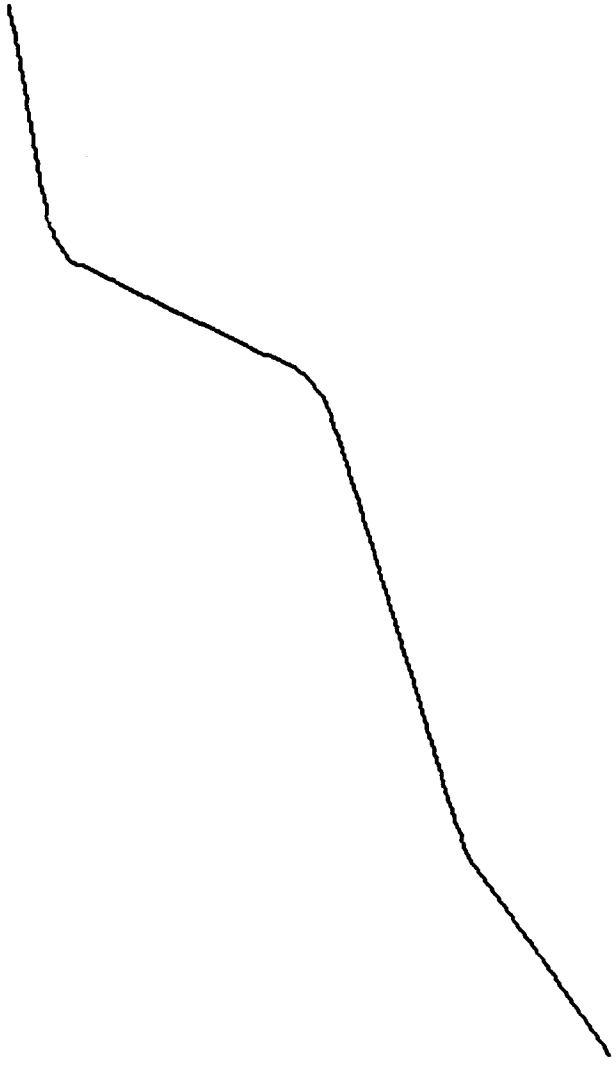


Figure 13. The railroad map.

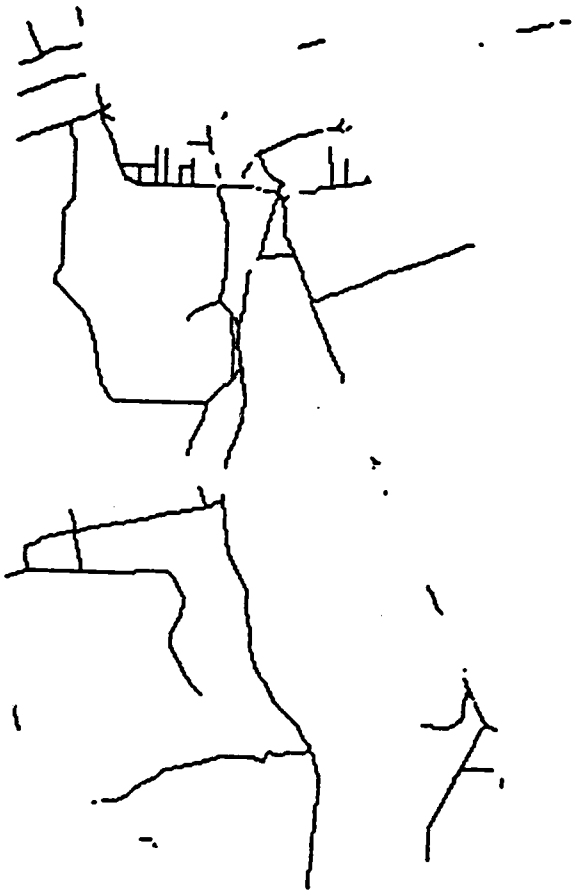


Figure 14. The map named 'lowroad' (the intersection of the road map with the map named 'low').

Figure 15. The house map.

Figure 16. Intersection of the house map with 'low'.

2.4. On the timing of the query language demonstration

The timing of the execution of commands in the query language, as done for Phase II, is quite different from the timing of the programs developed in Phase I. First, a different mechanism is performing the timings. In Phase I, a system routine was being accessed directly by the C program that was performing the calculation. In Phase II, the timing is being done by the LISP system. Therefore, in addition to the time it takes for the C functions to perform the query, our values include the time necessary for the LISP functions to interpret the query language, parse the command, and cross the interface between LISP and C. Second, the kernel (which was not used in Phase I) explicitly performs memory management functions that were handled less efficiently, but invisibly to the Phase I timing mechanism, by the operating system in Phase I. Third, the quadtree file representation used in Phase II requires considerably more space than the file representation used in Phase I, but the Phase I quadtree files had to be read sequentially, whereas the Phase II quadtree files can be accessed randomly. The result of this is that in Phase II, commands that manipulate a part of map have been speeded up at the expense of commands that manipulate the entire map.

The main intent of timing measures is to give a feel for how interactive a process is. Timings are most useful when the process being timed is compute bound (as opposed to I/O bound). Unfortunately, it is not clear whether significant portions of the database system are compute bound; but given this and the above considerations, we submit the following two timing results.

The first timing task analyzes a typical work session involving a wide variety of queries. Table 1 presents the timings for the query language demonstration of section 2.3. In both this table and the later table, time is measured in seconds. Looking at the timings, we find that the most expensive operations are taking a single area map and constructing a new map that corresponds to a subset of the regions in the map or the windowing (clipping) of a region of the map. This is because these operations require the most processing time per node of the quadtree. For example, windowing requires calculation of what portion of each node lies within the window. The processing of a node to determine whether or not it is in a given class (as is done in the case of the construction of the Low Map) would be trivial except that it is done in a general setting where the node could conceivably be compared to many class or polygon types; thus carrying more overhead than is obvious from the usage given. The cost of the intersection commands are not surprising when one considers the size of the quadtrees involved (as shown in Table 2) in the case of area maps and

the necessary calculations performed per node in the case of the point and line map intersections.

The second timing task analyzes a single query (the intersection query) on a wide variety of data. In Table 3, results are shown for the intersection of each of the landuse classes (see Figure 3 and the Phase I report [Rose82a]) with each of the following three maps: the Center map (Figure 6), the Houses Map (Figure 15), and the Road Map (Figure 10). The size information given in Table 3 indicates the result of calculating the area of the map that results from the intersection. Note that the time needed to construct a map for each of the landuse classes is not included in these times. Recall from [Rose82a] that the landuse class ws was the Russian River - Dry Creek water system of Figure 1. Then, looking at Table 3, we see that 9 houses are inseparable from the water at the level of accuracy (discussed in [Rose82a]) with which the maps are encoded.

Table 1. Timings for Example Demonstration
(time is measured in seconds).

Abbreviated Command	Time
Please <system>	0.1
Please <command>	0.1
Enter flood	0.0
Enter land	0.0
Let be denote denote	0.1
Let extra be land	0.1
Describe extra	0.0
Let x be 100	0.1
Let y be 400	0.1
Let z be polygon x y extra	0.2
Use 0 0 512 512	0.1
Let zmap rename map z extra	10.3
Let center be map class 100 400 flood flood	10.2
Display center	1.4
Let left rename center	0.1
Display center - error return	0.0
Display left	1.2
Let frame be smallest zmap	1.0
Enter top	0.0
Let zz be windowing land frame	82.1
Display zz	1.0
Use 256 256 128 128	0.1
Display zz	2.1
Use 0 0 512 512	0.2
Display value area left	2.1
Display value perimeter left	10.0
Let center be map class 100 300 flood flood	13.2
Display center	1.9
Let low be map level1 top	34.3
Display low	1.7
Let step1 be intersection land low	67.0
Display step1	5.9
Let final be intersection step1 center	38.2
Display final	3.7
Enter road	0.3
Display road	3.8
Let lowroad denote intersection road low	30.0
Display point	1.0
Display intersection point low	5.5

Table 2. Sizes of Maps Referred to by Table 1.

Name of Map	Number of Nodes in Map
center	4687
extra (land synonym)	28447
final	10090
flood	5248
intersection point low	454
land	28447
left	3197
low	4996
lowroad	3091
point	1906
road	7729
step1	15898
zmap	1765
zz (shifted zmap)	3805

Table 3. Timings for Intersection Task
Intersection of each class from the landuse map with:
1) the center region of the floodplain map (an area map)
2) the house map (a point map)
3) the road map (a line map)
(Note: size is measured in number of non-white pixels
time is measured in seconds)

Land Class	Center Map Time	Center Map Size	Houses Map Time	Houses Map Size	Road Map Time	Road Map Size
acc	13.4	6341	2.7	9	13.9	30
acp	19.3	26886	3.1	25	15.8	94
ar	5.9	1197	2.6	11	7.9	12
are	4.3	152	2.4	0	5.8	0
avf	31.7	23776	3.7	59	29.5	264
avv	32.9	29685	3.1	50	28.6	341
bbr	5.0	432	2.4	0	6.2	0
beq	4.6	229	2.4	0	6.3	0
bes	4.2	147	2.5	0	6.4	0
bt	8.7	3403	2.9	13	8.5	3
fo	16.4	16952	2.5	4	11.7	30
lr	7.0	948	2.4	0	7.3	1
r	19.2	23147	2.5	4	14.6	94
ucb	4.4	249	2.4	0	6.8	14
ucc	5.5	1018	2.6	4	8.4	34
ucr	6.3	1518	2.5	1	9.3	90
ucw	4.8	305	2.6	2	6.9	4
ues	6.6	1628	2.4	1	9.0	33
uil	4.8	422	2.6	0	7.7	13
uis	6.1	1042	2.6	6	8.8	18
uiw	4.6	186	2.8	4	6.8	6
uoc	4.2	288	2.6	0	6.5	14
uog	4.5	1115	2.6	2	7.0	30
uoo	4.7	490	2.5	1	7.1	11
uop	4.4	213	2.8	10	7.1	8
uov	4.5	238	2.6	9	6.2	6
urh	4.5	167	2.5	3	6.6	3
urs	26.9	26752	10.6	577	46.5	1098
uus	4.8	261	2.5	0	6.4	0
uut	10.4	1928	2.6	2	12.7	18
vv	4.2	108	2.3	0	6.1	2
wo	4.3	0	3.0	0	6.9	0
ws	13.7	3409	2.7	9	15.3	11
wwp	4.6	206	2.5	0	6.0	0

3. The quadtree editor - a tool for database update

3.1. An overview of the quadtree editor

The quadtree editor (QED) exists to facilitate the interactive construction and updating of maps stored as quadtrees. Rather than forcing the user to think in terms of the tree structure, QED tree manipulation commands make references to logical units of the map (e.g., lines, points or polygons). The user performs editing operations such as inserting a line or point, changing the value of a specified polygon, or splitting a specified polygon into more than one piece.

When many changes are to be made, the user may wish to see the effects of each step. Commands are provided to allow him to examine all or part of the map at a selected location on a display device. This display is continuously updated as further map manipulation commands are executed. Associated with each map's quadtree representation is a descriptor termed the quadtree header. It contains information such as the size and location of the map. There exist commands which allow the user to modify this header. In addition, he may also insert textual comments into the header for documentation purposes.

QED is a command based system - i.e., the user gives a command and it is executed, after which the system is ready to execute the next command. There is no notion of composing functions as there is in the quadtree database language. When the editor is ready to receive a new command the prompt <?> is displayed. Area maps are updated by use of the replace, change, and split commands which replace all polygons of a given value with a new value, change the value of a given polygon, or split a polygon into multiple polygons, respectively. Line and point maps are updated by use of the insert and delete commands which insert or delete lines or points, respectively. In order that the user may see what he is editing, there are commands that draw all or part of the map onto a selected section of the Grinnell display device. The user may also alter the header of the map.

To begin using the editor from the database system, the user types

Edit <file> please

where <file> is the name of the disk file in which the map to be edited is stored. If the file does not already exist, then the editor assumes a new map is to be created. In this case the user will be prompted to indicate what the map type is (region, line, or point data). New maps are initially entirely white. If the file does exist, then it must be a legal quadtree file; otherwise QED informs the user that the file cannot be edited and halts. In order to protect the

user against errors and machine crashes, a complete copy of the file being edited is made, and all editing is actually done on the copy. At the termination of an editing session the old copy of the file being edited is stored in a backup file, and the copy containing the revisions replaces it. For example, after editing a file named "mymap", the prior version is stored in the backup file "mymap.bk", and the new (edited) version will then reside in "mymap". The commands typed while editing are also recorded in a file with the suffix ".ty". Therefore, in this example, file "mymap.ty" would contain the commands used in the last editing session.

Sections 3.2 describes each command in greater detail. Where syntax lines are given, names enclosed in angle brackets <> are syntactic type indicators - the user would type a numerical value or name in their place. Arguments enclosed in square brackets [] are optional. In Section 3.3, some of the implementation details of QED commands for region maps will be considered. Implementation details for point and line maps may be found in Section 5. Section 3.4 presents a demonstration of database updating for an area map.

3.2. Quadtree editor commands

3.2.1. Header and comment commands

Each map file includes a header which contains information such as the width of the map, its location and orientation in space, and the type of data represented - say, topography or landuse data for region maps, roads for line maps. A comments section is also provided to allow the user to document his maps. The following commands allow the user to change the header, add comments, and read the header. Note that once the map's type has been set to region, line, or point, the map type can not be changed.

3.2.1.1. Header

Syntax: Header

This command allows the user to view and alter information in the header. For each item of data contained in the header, the editor gives the user the opportunity to change it. The editor outputs a description of the information, its old value (in parentheses) and a prompt for the user to insert the new value. If no change is desired, typing the return key will leave that item as is. The modifiable values are:

Map size - All maps are a square of size N x N where N is some integer power of two. If the value given is not a power of two, then it will be converted to the least power of two greater than the given value.

First X and First Y - The external coordinates of the lower left corner of the map. This might be used by other functions in the database for comparing the relative positions of two maps.

Rotation Angle - A real number which is the tilt (in radians) of the map from the external horizontal. Once again, it could be used to compare two maps.

Data Type - A single capital letter. This describes the type of information conveyed by the map. Some currently understood values are:

- B binary map (all polygons are black or white)
- T topography map
- L landuse map
- U unknown type

When creating a new map, the editor will automatically execute the header command in order to allow the user to give initial header information.

3.2.1.2. Comment

Syntax: comment <comment-line>

Add <comment-line> to the comments which are stored along with the header. These comments are provided by the user to say whatever he wishes about the map. Some database functions may also add to the comments. For example, when QED creates a map, that map receives a comment stating that it has been created by QED. All comments are automatically prepended with the date and time.

3.2.1.3. Print

Syntax: print

Print out the header and comments (see header and comment commands).

3.2.2. Grinnell manipulation commands

The following commands allow the user to select the section of the map he wishes to view and on what portion of the Grinnell. When the Grinnell viewing functions are on, all changes to the map which occur within the user's viewing window will be displayed.

3.2.2.1. Gon

Syntax: gon <fx> <fy> <nx> <ny>

This command selects (or changes) the section of the Grinnell on which the user will display his map. The four integers <fx>, <fy>, <nx> and <ny> describe the window - lower left x and y coordinates, width and height, respectively (the lower left corner of the Grinnell is assumed to be (0,0)). The selected section of the Grinnell is erased by this command.

3.2.2.2. Goff

Syntax: goff

This command releases the Grinnell. After this command is given the Grinnell window is erased, and no further changes to the map will be shown on the Grinnell.

3.2.2.3. Look

Syntax: look <fx> <fy>

This command selects the portion of the map which is to be viewed, and displays it on the Grinnell in the window defined by the last gon command. The integers <fx> and <fy> describe the lower left x and y coordinates of the portion of the map to be viewed (the height and width are taken from the previously defined Grinnell window).

3.2.2.4. Point

Syntax: Point <x> <y>

This command places a flashing cursor in the Grinnell window at the point specified by the arguments.

3.2.3. Area map changing commands

The three commands replace, change and split enable the user to make changes to a region map. These changes are reflected on the Grinnell if they occur within the current user specified window of the map and a Grinnell window has been opened by a gon command.

3.2.3.1. Replace

Syntax: replace <old-val> <new-val>

All polygons of the map with (integer) value <old-val> will be replaced by polygons of type <new-val>.

3.2.3.2. Change

Syntax: change <x> <y> <new-val>

This command changes the value of one particular polygon of the map to the given (integer) value <new-val>. The integer values <x> and <y> define a coordinate which lies within the polygon to be changed.

3.2.3.3. Split

Syntax: split [s] <val> [<file>]

This command is used to split a polygon into more than one region. The user supplies a chain code which is drawn onto the map with value <val>. Typically, one of the resulting subpieces of the polygon will subsequently be given a new value with the change command.

The chaincode may be entered in one of two ways. If <file> is given, then the editor gets the code from a disk file with the name <file>. Otherwise, the editor will

prompt the user to input the chaincode online. The syntax of a chaincode is as follows:

(<x>,<y>)<list>#

<x> and <y> specify the beginning coordinate. Each direction is either one of the letters h, j, k or l, or else a number followed by one of those letters. The letters have the following meanings:

h	move left
j	move up
k	move down
l	move right

If a number is given before the letter, than the code moves that number of pixels in the appropriate direction. An example of a chaincode starting at (100,100) and forming a 5 X 5 box would be:

(100,100)4l4k4h3j#

As the user types the chaincode (or as it is read from the file), it is drawn onto the map. It is also displayed on the Grinnell if a window has been opened, thereby enabling the user to see the chain growing as he inputs it. Typing the backspace key will erase the last pixel of the chaincode. If the "s" option has not been given then the chain may extend across any number of polygons. If the "s" option is given then the chain will stop when it attempts to cross into a polygon other than the one it began in. In either case, the chain will stop if it attempts to go off the edge of the map.

3.2.4. Point map changing commands

The commands insert and delete enable the user to make changes to a point map. These changes are reflected on the Grinnell if they occur within the current user specified window of the map and a Grinnell window has been opened by a gon command.

3.2.4.1. Insert

Syntax: insert <x> <y>

A single point is inserted at the coordinates specified by <x> and <y>. If a point already exists there, nothing will be changed.

3.2.4.2. Delete

Syntax: delete <x> <y>

The point at the specified coordinate is removed. If no point exists there the editor will complain, but nothing will be changed.

3.2.5. Line map changing commands

The commands insert and delete enable the user to make changes to a line map. These changes are reflected on the Grinnell if they occur within the current user specified window of the map and a Grinnell window has been opened by a gon command. While they have the same name as similar commands for point maps, they take different parameters.

3.2.5.1. Insert

Syntax: insert <ax> <ay> <bx> <by>

A line segment is inserted from (<ax>,<ay>) to (<bx>,<by>).

3.2.5.2. Delete

Syntax: delete <ax> <ay> <bx> <by>

A line segment is deleted from (<ax>,<ay>) to (<bx>,<by>). If no line segment exists over all or part of this span, no change will occur in the clear sections. If, however, another line segment crosses the (non-existent) segment specified by the user, this other line segment may develop a gap where the intersection occurs. The user is encouraged to view line segments as atomic units when deleting. Only segments known to have been inserted should be deleted, and the end points should be given in the same order. The safest way to delete a part of a line segment is to remove the entire line segment and then re-insert the appropriate parts. This way one avoids problems relating to roundoff errors in line slope calculations.

If used correctly, where the deleted line segment passes through another line segment, this other line segment will not be left with a gap.

3.2.6. Miscellaneous commands

3.2.6.1. Quit

Syntax: quit

This command signals the editor to save the changes made to the map and finish processing. It is the normal way to exit the editor. When this command is given, QED will ask the user for a parting comment. This makes it easy for him to keep a history of editing sessions on the file. If no comment is desired, typing a carriage return will insert no comment. Otherwise, the user's comment and the date will be inserted into the comments section.

3.2.6.2. Abort

Syntax: abort

This command signals the editor to stop without saving the changes to the map. The state of all files is exactly as it was before the editing session took place, except that the file which contains the commands typed during an editing session will reflect this latest session. This command protects the user in case of error.

3.2.6.3. Shell

Syntax: shell <command>

This command enables the user to access the UNIX shell. Whatever the user types as the <command> argument will be executed as though the user were not in the editor. After the command is executed, the bell on the terminal will ring, and the command prompt will be given.

3.2.6.4. Help

Syntax: help [<command>]

If <command> is not specified, then a list of the editor commands is displayed along with their syntax (as given on the syntax line in this section) and an extremely brief description of their functions. It serves to remind the user of the available commands and their correct usage. If <command> is given, then a longer description of that command is output on the user's terminal. The help command (as well as the editor itself) is controlled by the map type. This means that the commands which the user sees relate to the map currently being edited. If, for example, the map being edited is a line map, help with no <command> option will give the list of commands applicable to line maps. The change, replace and split commands will not be listed. The insert and delete commands will each be listed with four parameters for specifying line segments. Both asking for help or trying to execute a change command while editing a line map will result in the editor complaining that this command does not exist. Asking for help on the insert command while editing a point map will give the description in

Section 3.2.4.1.

3.2.6.5. Value

Syntax: value <x> <y>
Alternatively: value -

This command returns the value or color of the node at a given pair of coordinates in the map. If "-" is the only argument, then the coordinates are taken from the position of the Grinnell cursor. This enables the user to determine the point visually via the trackball.

3.2.6.6. Set

Syntax: set <variable> <value>

This command allows the user to change the value of certain user accessible variables. The only variable implemented in the editor is called "global." It may be set to either "true" or "false" and is initialized to "false." So, to use this command to change "global," the user would type:

set global true

When global is set to "false" all coordinate values supplied to commands are interpreted relative to the lower left corner of the map being edited. When global is set to "true" all coordinate values are interpreted relative to the global coordinate system - in other words, the editor looks at the values for First X and First Y as set by the header command. These values are subtracted from the given coordinate to calculate the position relative to the lower left corner of the map. Coordinates returned by editor commands will also be relative to the global system when global is set to "true," and relative to the lower left corner of the map when global is "false."

3.3. Implementation of the quadtree editor functions

As explained in Section 4, all quadtree primitives and memory management functions are handled by an underlying set of general purpose quadtree routines. The quadtree editor itself views these functions as atomic actions. The editor proper is then concerned with receiving and executing commands provided by the user.

When the editor is called, the user gives the name of the file to be edited. A temporary disk file is created on which all editing is to be done. Another file is created to store the commands given by the user. These files help protect the user from serious loss due to system crashes or his own errors such as mistyped or unwanted commands. They also enable him to abort the editing session without damaging the original copy. If the file to be edited is an old one, a copy is made in the temporary file. If a new map is to be created, then a default header is installed and the map is initialized as all white. The memory management initialization routine is executed and (for new maps) the user is requested to supply initial header information through the header editor function.

From here on, the editor accepts commands from the user. For each command, it parses the command line and executes the request. Many of the commands were quite easy to implement. The abort command needs only to remove the editor's temporary file (since the original file and its backup are not affected by the editor). The quit command first invokes the memory management functions which write the quadtree and memory management headers to disk along with the B-tree pages currently in core, and then moves the original quadtree file to the backup file and copies the temporary (edited) file in place of the original. The header command changes values in the header structure. The comment command passes the user's comment to the memory management function which inserts the comment into the comment list. Print uses memory management functions to read comments and then writes them with the header to the user's terminal. The shell command uses a standard C system call to allow the user to execute normal C program calls.

The Grinnell accessing commands gon and goff change global variables used by the map manipulating functions. These variables include a Boolean flag to indicate if map changes are to be displayed, and a description of the Grinnell window available. The look command (after changing other global variables which describe the map window) performs a traversal of the entire tree. For each node, it determines if the node lies within the chosen map window. If it is, then the appropriate offsets are calculated and the node is displayed on the screen.

The commands value and point were included to assist the user in relating the map coordinates to what is displayed on the Grinnell. Without them, choosing coordinates for the beginning of a chaincode or determining the current value of a polygon would be difficult. The value command searches for the node at the requested location and returns its value. The point command uses a standard Grinnell primitive function to set the cursor to a requested position.

In order for the quadtree editor to be useful, a set of map manipulating functions is needed that permits the user to create any desired map. The user of a geographic database system such as this will view the units of his map in terms of logical units such as "lines" or "polygons," and not square "nodes." Therefore, for region maps it is clear that the most natural implementation is one that allows the modifying commands to make changes to specified polygons. This means that when implementing these commands it must be possible to modify all of the nodes which make up a polygon or group of polygons without affecting nodes of neighboring polygons. In our system, each node has a value field. Each polygon (and hence each node making up the polygon) is considered to be a member of a "class". This class could be an elevation range or a landuse type such as "wheatfields". The value of the node indicates the class of which it is a member.

The following commands apply only to region maps. Inserting and deleting lines or points is discussed in Section 5.

The replace command is executed by traversing the entire quadtree. Those nodes with the old (class) value have that value replaced by the new. For this command it is not necessary to distinguish between polygons of the same class since they are all processed in the same way.

The change command is more complicated. This command should manipulate only one polygon; however, other polygons of that class may also exist. After the command line has been parsed, a recursive function is called which actually performs the desired work. This function takes a node as its parameter. This node is checked to see that it has the old value (the one to be changed). If so, then its value is changed to the new one and the function is recursively applied to all of the node's neighbors. In this way, all nodes of a polygon will eventually be reached and only nodes in the polygon will have their values changed (since only four-neighbors of nodes in the desired polygon are ever candidates for processing).

The split command allows the user to impose an arbitrary line, one pixel wide, of a designated value onto the map. The intended use of the command is to split a polygon into two or more separate parts. One of these parts would then become a polygon of the same class as the chaincode via subsequent invocation of the change command. The pixels of the chaincode would then be part of this new polygon. Alternatively, the split command can be used to make slight modifications of only a very few pixels, such as correcting a slightly misplaced border of a polygon. This type of correction could not be applied in any other way with the available command set.

The split command operates by first inserting a one pixel node into the tree corresponding to the first coordinate given and then following the chaincode inserting nodes as it reads the code. As the user types the code, the code is also inserted into the command file. Allowing the user to observe the progress of the chaincode as he is inputting it is a key feature of our implementation of the split command. Typing an incorrect chaincode was judged to be a very common source of error when the implementation was designed. Enabling the user to see the chain displayed as he inputs it allows the rapid detection and correction of errors. When the backspace key is typed the chaincode is backed up one pixel. This is accomplished by examining the end of the command file. The last direction of the code is read to determine the coordinates of the previous pixel of the chaincode and then both the map and the command file are updated to reflect the backup. In typical usage of the split command, the result will be a line splitting a polygon into two or more pieces. The user would then change the value of some of the pieces via the change command.

By repeated use of the three commands replace, change, and split, it is possible to a make any desired changes to a region map. Clearly this is true since in the worst case the user could construct an entire map from one pixel chaincodes. However, it is hoped that the provided commands are of sufficient power to enable a user to easily edit maps as he wishes.

3.4. A demonstration of database updating.

Figure 17 shows the original floodplain data provided for this project. Also included are four regions bordered by dotted lines indicating intended revisions to the landuse map. This section demonstrates how these revisions can be executed using the quadtree editor. When in the database system, if the user wishes to edit the landuse map (which has been given the name 'land') he would begin by typing:

Edit land please

The editor will then start up, prompting the user with the symbol <?>. Following are the necessary commands to create Figure 19 from Figure 18, along with explanations of the steps taken. Messages from the editor to the user are typed in capital letters.

```
OLD FILE
<?> gon 0 0 512 512
<?> look 0 0
DISPLAY AREA MAP
28447 NODES DISPLAYED
```

The editor informs the user that the file is an old (already existing) quadtree file. The gon command clears the Grinnell screen, then the look command displays the landuse map for the user to see. There are 28447 nodes in this tree.

The map of revisions shows a polygon which should have class value of ACC. In the original landuse map, this area is two polygons - the one on the left having value AVV, and the one on the right having value ACC. The change command is used to convert the AVV part to ACC.

```
<?> change 19 247 100
# NODES FOUND: 408, # NODES IN POLYGON: 112
```

This command changes the color value of the polygon containing the point (19,247) to color 100 (the integer code for landuse class ACC). During the processing, the change function examined 408 nodes, of which 112 where actually in the polygon and had their value changed. The AVV polygon has now merged with the ACC polygon to create the desired revision.

The AVV polygon is somewhat harder to construct. On the original landuse map, this region contains a polygon of class AVV and a polygon of class ACP. The rest of the new region consists of two parts of a large AVF class polygon. To make the requested revision, it is necessary to draw the border of the new polygon through the AVF section. This is done with the split command in two pieces. The newly creat-

ed polygons will then be converted to class AVV via the change command. Lastly, the ACP polygon will be converted to AVV.

```
<?> split 96
PLEASE ENTER CHAINCODE (ENDING WITH #):
(32,238)klklklkl11k#
<?> split 96
PLEASE ENTER CHAINCODE (ENDING WITH #):
(62,219)4khkh7k7hj10h#
```

Note that 96 is the integer code for AVV and that the line is therefore of class value AVV. The following change commands complete the AVV polygon revision.

```
<?> change 50 211 96
# NODES FOUND: 125, # NODES IN POLYGON: 33
<?> change 29 227 96
# NODES FOUND: 288, # NODES IN POLYGON: 74
<?> change 32 207 96
# NODES FOUND: 265, # NODES IN POLYGON: 70
```

The UIS polygon is formed in a manner similar to that used to form the AVV region - once again the split command is used to complete the border of the new polygon, and the change command then merges the pieces together. 3080 is the integer code for UIS.

```
<?> split 3080
PLEASE ENTER CHAINCODE (ENDING WITH #):
(53,273)32h10jljjh4jljjl21j#
<?> change 30 297 3080
# NODES FOUND: 321, # NODES IN POLYGON: 82
<?> change 26 289 3080
# NODES FOUND: 94, # NODES IN POLYGON: 27
<?> change 25 279 3080
# NODES FOUND: 99, # NODES IN POLYGON: 27
```

For the final revision, the URS polygon, we simply draw the complete border of the polygon and fill in the pieces. 3268 is the integer code for URS.

```
<?> split 3268
PLEASE ENTER CHAINCODE (ENDING WITH #):
(213,205)68l15kl5kllkllkkhkkl10k3h15k45hk6h36jh9j19h6j#
<?> change 241 203 3268
# NODES FOUND: 559, # NODES IN POLYGON: 151
<?> change 240 179 3268
# NODES FOUND: 217, # NODES IN POLYGON: 63
<?> change 253 199 3268
# NODE FOUND: 139, # NODES IN POLYGON: 41
<?> change 264 188 3268
# NODES FOUND: 283, # NODES IN POLYGON: 79
```

```
<?> change 260 162 3268
# NODES FOUND: 232, # NODES IN POLYGON: 63
<?> change 276 190 3268
# NODES FOUND: 272, # NODES IN POLYGON: 77
```

The line codes for this demonstration were of course prepared beforehand, and the commands shown here are polished final results. However, a user who has only a hard copy map in front of him can determine by eye what points to use via the point and value functions. He can easily correct errors while drawing line boundaries by using the backspace key. So while this demonstration does not show commonly occurring errors, these errors can easily be dealt with by the user as he makes them.

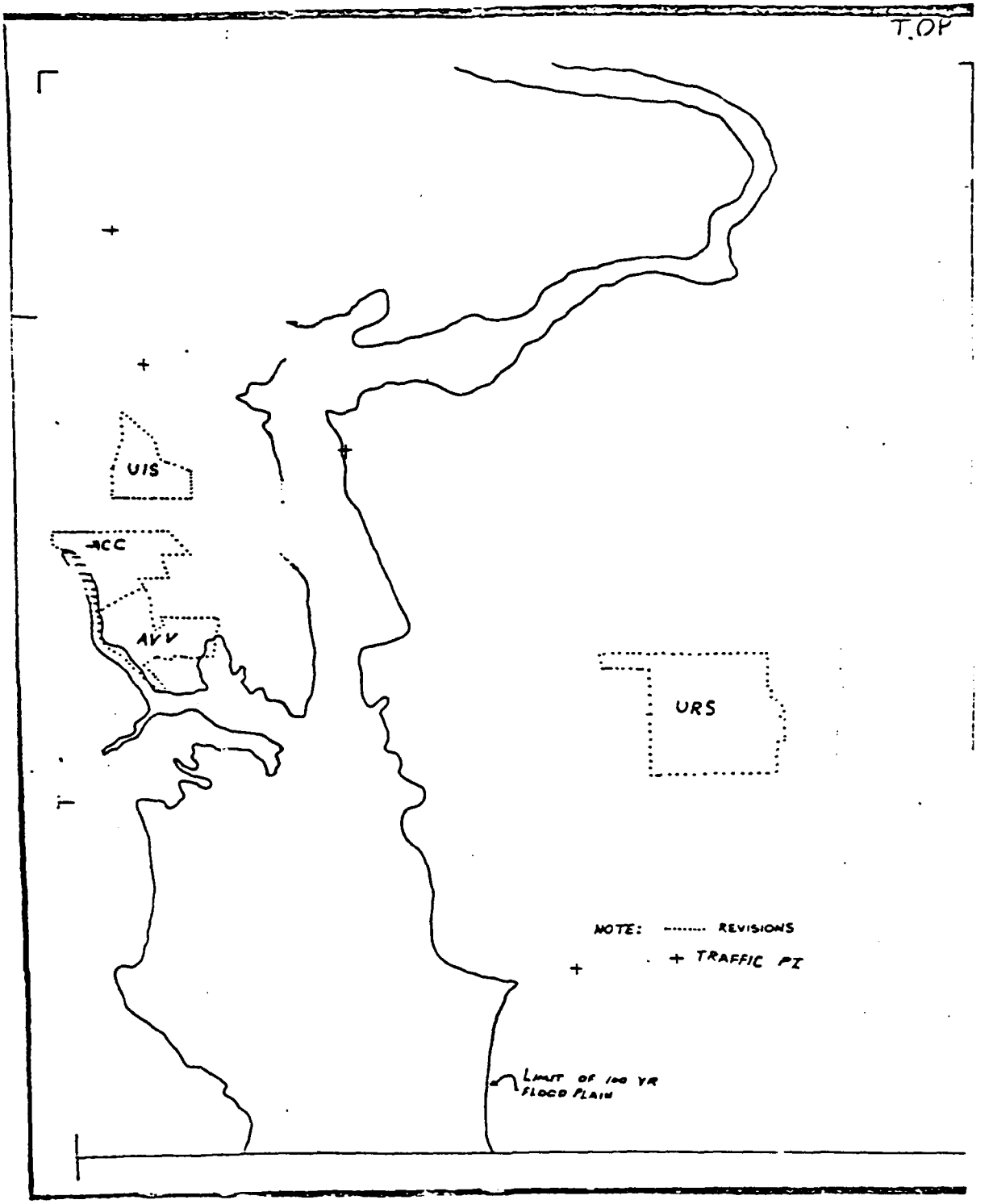


Figure 17. The floodplain map with marked revisions

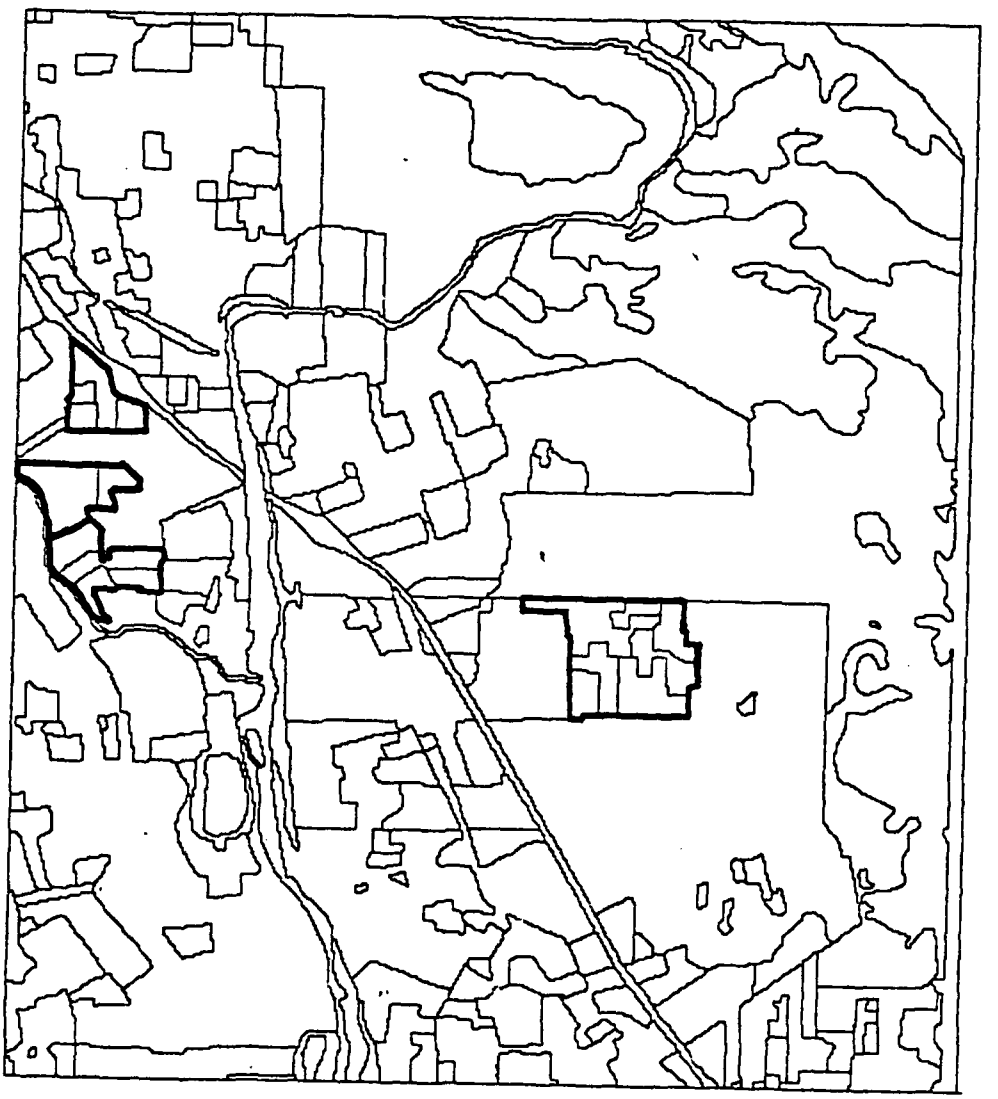


Figure 18. The landuse map.

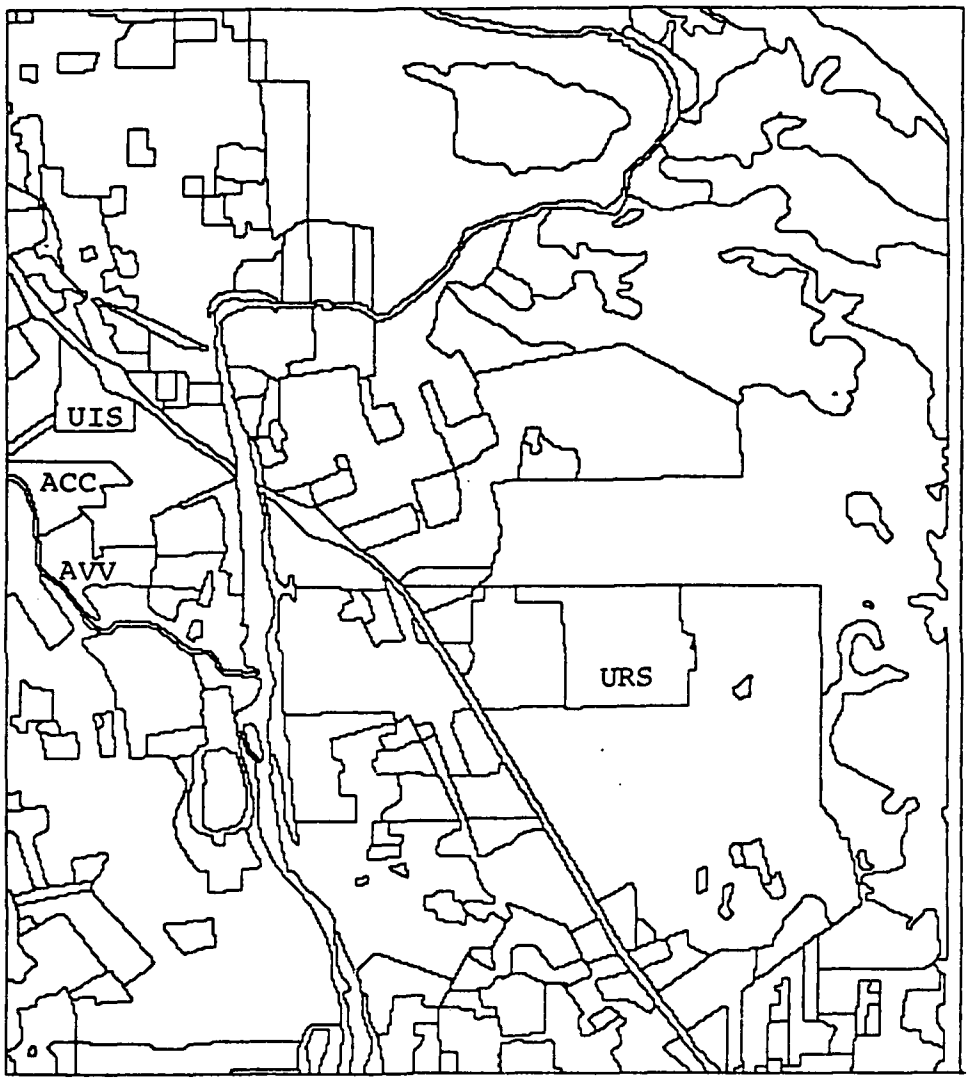


Figure 19. The updated landuse map.

4. The quadtree memory management system

4.1. The user's view of the memory management system

The quadtree memory management system, henceforth known as the kernel, controls the interface between the quadtree files and the programs that the database uses to view and manipulate quadtrees. At the level of the kernel, the three types of quadtrees (region quadtrees, point quadtrees, and edge quadtrees) are identical. The kernel views the quadtree file as a collection of quadtree leaves, each leaf having some value associated with it. One could view the kernel as defining the quadtree file as an abstract type, as all routines in the database use the kernel to access the quadtree files. The word 'user' in this section refers to the programmer of the quadtree database (i.e. the 'user' of the kernel routines), not the 'user' of the quadtree database system described in Section 2.

From the user's point of view, a quadtree file has three parts: an array of user defined information, a list of comments, and a list of quadtree leaves.

The array of user defined information, known as the user's header, is a fixed size array that is set up when the quadtree file is created and its usage is totally up to the user. The kernel supports the array by offering the user two routines: read_head and write_head. These routines transfer the user's header between the quadtree file (where the user can't change it, but it is associated with the file) and a character array (where the user can change it, but it is no longer associated with the file). The routines read_head and write_head do not allow access to any part of the quadtree file that is outside of the initially defined user's header. Typical usages of the user's header would be to keep track of whether a quadtree file is to be interpreted as a point quadtree or a region quadtree, the x and y coordinates of the lower-left-hand corner of the quadtree (with respect to some global coordinate system), and how many pixels wide the quadtree is.

The list of comments provides a variable size disk area where the user can place comments (usually a list of the function calls used in the creation of the file). The commenting feature is maintained by three routines: append_cmnt, read_cmnt, and cmnt_init. Note that a comment cannot be changed once it has been inserted, because the only write capability is to append. The read_cmnt routine allows the user to fetch the next n characters of comments. Thus it was necessary to provide a cmnt_init routine, to return the read routine to the beginning of the comment list (allowing rereads).

Most of the code in the kernel is dedicated to maintaining the list of quadtree leaves. The kernel user interfaces with this code through twenty C subroutines and macros. The major distinction between these routines is whether they access the quadtree file or are utilities for manipulating quadtree leaf descriptions.

The quadtree leaf description has three parts. It contains the depth of a leaf (accessed via qd_depth), the coordinates of the lower left hand corner of the leaf (accessed via qd_x and qd_y), and the user interpreted value of the leaf (accessed via qd_value). A leaf also lies in a particular quadrant of its father; which quadrant the leaf lies in can be computed by qd_which. As well as interpreting existing leaves, there are routines that allow the construction of leaf descriptions. The basic constructor is qd_clear, which creates a leaf of depth 0, with value UNUSED, whose x and y coordinates are <0,0> (which is always measured from a leaf's lower left hand corner where the entire tree's lower left corner has <0,0> for its coordinates). The user interpreted value of a leaf description can be changed using the qd_set command. A leaf description of a leaf having a certain depth and x and y coordinates relative to a tree with a given maximum depth is built by the qd_xy routine. A leaf description for a node which would be the father or son of a given node can be built using qd_father or qd_son (respectively). A copy of a leaf description can be made using the qd_copy routine. Two leaf descriptions can be compared relative to their visitation order by a preorder traversal ordering using the addr_gt function. The number of bytes used by a leaf description is kept in a macro called B_ADDR_SIZE.

Those routines that access the quadtree file can be divided into those that search the file and those that change the file. The most basic of the functions that search the file is qd_find. Given a leaf description, it returns the description of the leaf in the quadtree that would contain the given leaf. Sometimes the leaf to be found is larger than the actual leaves at the appropriate position in the file, there is no 'containing' leaf. In this case, qd_find returns the leaf in the quadtree contained by the given leaf's description that is least (using the ordering defined by addr_gt). If one wants to find a leaf description in the quadtree that is the preorder successor, neighbor with respect to a particular side, or a diagonal neighbor, then one uses the qd_preorder, qd_neighbor, or qd_diagonal functions (respectively). Often one wishes to apply the same function to every leaf in a quadtree. One way to implement this would be to have the function be the body of a loop that calls qd_preorder until it runs off the end of the map. However, qd_preorder works by first calculating the value of the successor of a given leaf, and then searching for this

newly calculated leaf. Since we know that we wish to execute the same function on every leaf in the tree in any convenient order, it is not necessary to search for a particular leaf. Instead we can use whatever leaf is next in the tree. Qd_travel is a kernel routine which takes a function as its argument, and applies this function to every leaf in the file in the most efficient manner.

The principal way to change a quadtree file is to insert a leaf using the qd_insert routine. If one inserts a leaf description that is identical to one that is already in the quadtree (except for the leaf having a different color), then the color associated with that leaf description is changed and if that causes any leaves to merge (due to four siblings having the same color), then those leaves are merged. If the leaf description to be inserted describes a leaf that contains more than one leaf already in the quadtree, then the contained leaves are deleted and the new leaf is inserted. Thus one can empty a quadtree by inserting a white leaf at depth zero. If one inserts a leaf description that is contained by a leaf description that is already in the quadtree, then the leaf in the quadtree is split into its four sons (each with the same color as the father) and the insertion attempt is repeated. There is also a quadtree file changer called qd_packer that tries to compress the arrangement of leaves in the quadtree file. Although qd_packer can change the quadtree file, its effects are invisible to the routines described above (with the exception that most routines run faster on a quadtree file that has been packed).

Finally, there are two routines needed to monitor the relation between the external operating system, in which the quadtree files persist across many invocations of the database system, and the kernel, whose memory vanishes when the database system exists. The first routine is qb_init, which allocates and initializes a portion of core that can be used by the routines that manipulate a particular quadtree file. The second routine is qb_post, which deallocates the portion of core that was associated with a quadtree file (most importantly, qb_post makes sure that all buffers have been written back onto the quadtree file).

4.2. Implementation of the memory management system

The major question in the implementation of the kernel is to decide how to order the storage of the leaf descriptions. It can be seen [Garg82a] that if one orders the leaves according to how they would be visited by a preorder traversal of the quadtree, then the next leaf description sought will tend to be near the last leaf description found. In the kernel, a leaf description is stored in two long words (32 bits each). The first long word is split into one field of 3 bytes (24 bits) that contains the result of interleaving the bits of the x and the y coordinates of the leaf described. The second field of 1 byte (8 bits) contains the depth of the leaf described. It can be shown that if one compares this four byte address descriptor of two leaves (using an arithmetic comparison of the absolute value of the two long words), then the leaf corresponding to the greater value will be visited later by a preorder traversal of a quadtree than will the other leaf. Thus we have a quick and efficient way of determining a linear ordering of leaf descriptions. The second long word of the leaf description is used to store the value/color of the leaf. Note that this leaf description structure will support quadtrees with leaves at depth twelve or less. This limit arises from the 3 bytes (24 bits) that are used to store the interleaved coordinates. This gives a maximum size of 2048 by 2048 pixels for any map. Our original database consists of three maps each approximately 400 by 450 pixels, so this is quite sufficient.

Given the linear ordering of leaf descriptions described above and the fact that we will be storing collections of leaf descriptions containing as many as 30,000 leaves, it is obvious that the quadtree files should be organized as some kind of b-tree structure [Come79]. The kernel thus becomes a collection of routines that maintain a buffer pool in core and a b-tree in the disk file, allowing the user to manipulate the leaf descriptions without having to worry about any of the details. Note that the buffer pool contains that portion of the quadtree file that there is room for in core at any given time and, due to locality of reference considerations, is probably best maintained on a schedule that replaces the least recently used buffers first.

We are currently using 512 byte b-tree nodes, which allow room for up to sixty leaf descriptions. Each b-tree node (except the root of the b-tree) is guaranteed to have at least thirty leaf descriptions in it. Thus one will rarely build a b-tree that is deeper than four b-tree nodes. There is much research that can be done with regards to the kernel implementation. It would be interesting to know the effects of different size b-tree nodes, different size

buffer pools, and different b-tree node balancing schemes on the response time of the kernel.

5. Point and line data

In the final report on Phase I [Rose82a], the details of implementing region quadtrees were discussed. In Phase II, quadtrees were also used to store two new types of data: points and lines. The details of our usage of quadtrees for these two new data types will be discussed below. It should be noted that the same kernel (described in Section 4) is used for manipulating quadtrees involving each of these three data types. When storing area data in region quadtrees, the value of a leaf corresponds to the color of the region that contains the leaf. Since there is no notion of color associated with either point or line data, other interpretations will be placed on the information stored in the value portion of the leaf description. What interpretation a particular routine makes of a leaf's value is dependent on what type of data is being stored in the quadtree. The user keeps track of this in the user header described in Sections 4.1 and 3.2.1.

To be precise, the routines that manipulate region quadtrees view the 32 bit value field associated with each leaf's location as containing three subfields. The first field (leftmost bit) is set to indicate that an erroneous value has been placed there. This corresponds to using a negative long integer as the value of a node. As it happens, the routine that creates an empty leaf description (qd_clear) places a -5 in the value field to indicate that no one has specified a value for this leaf description yet. This is consistent with the usage described, as something is probably wrong if there is a leaf description in the tree that does not have an assigned value. The second field (also a one bit field) is the mark bit that is used by sweep-and-mark type algorithms, e.g., connected component labeling. The remaining field (30 bits) contains the color of the described leaf.

The implementation of the point quadtree interprets the value field as containing five subfields. The first two fields are the error and mark fields that are used just as in the region quadtree. The third field (two bits) is unused. The fourth field (14 bits) contains the x-coordinate of the point stored in the leaf. The fifth field (also 14 bits) contains the y-coordinate of the same point. Fourteen bits is more than sufficient considering the implementation restriction of using quadtrees with a depth less than or equal to twelve. Note that a depth of twelve will handle a 2048 by 2048 map. Thus we can interpret a y value of 4096 as an unused value that can be used to denote a leaf description for a region that contains no points.

The above interpretation of the leaf description value field has the following consequences with respect to point quadtree algorithms. No more than one point can be stored in a quadtree leaf. Insertion of a point in a point quadtree works as follows. First we find the leaf that contains the point's location. If the leaf is empty, then the point's x and y coordinates are placed in the leaf description. Otherwise, the leaf is split into its four sons, the old leaf's point value is copied into the appropriate son, and insertion is re-attempted. Deletion of a point in a point quadtree is a matter of finding the leaf that contains the point and then changing the leaf description to that of an empty leaf. Next, one checks to see if it is possible to merge the new empty leaf with its siblings.

The point quadtree described above differs from the original point quadtree of Finkel and Bentley [Fink74], in that the structure of our point quadtree is independent of the order of point insertion. This is a result of the fact that leaves are always split by the kernel into four congruent squares, whereas the original point quadtree split leaves up into rectangles whose dimensions were a function of the first node inserted into the leaf's region.

The value field of the line quadtree leaf description has four subfields. The first field (one bit) indicates error values as it does for the other two types of quadtree leaf descriptions. The second field (one bit) indicates whether or not the leaf corresponds to a single pixel in the map. The third field (two bits) tells which son a node is with respect to its father. By setting this field, we guarantee that the leaf will not be automatically merged with its brothers by the kernel's insert routine. The only time this field is not set is when the leaf contains no line segments. Thus empty regions automatically merge.

The fourth field (28 bits) of the line quadtree interpretation of a leaf's value contains different information depending on whether or not the leaf corresponds to a single pixel in the map. If the leaf corresponds to a pixel, then the fourth field indicates how many lines pass through that pixel. If a leaf corresponds to a larger region, then it is either empty (and contains a special empty leaf code) or it contains exactly one line segment. If it contains exactly one line segment, then the intercepts of the line segment with the leaf's region are stored. Since the line must intercept the region at the region's perimeter, it is possible to encode each intercept with 14 bits and be able to handle regions as large as 4095 by 4095. In our implementation, this fourth field is further split into four subfields. The first two bits determine the side of the node on which the first intercept occurs. The next 12 bits indicate how far from the corner the intercept occurs

(the left corner for the north and south edges, the lower corner for the east and west edges). The next fields of two and 12 bits repeat this for the second intercept.

The insertion and deletion algorithms for line quadtrees are designed so that if one viewed the line segment as an indivisible atomic unit, then line segments could be dynamically inserted and deleted without roundoff errors creeping into the map representation to the extent that a line's endpoints have changed. This is important because the only way of indicating a line is by specifying its endpoints. Note however that the representation does not explicitly store the endpoints of a line segment, but rather stores a compact form of the digitization of all the lines in the map. The digitization of the lines is eight-connected, i.e., connection of the line is maintained across horizontal, vertical, or diagonal neighbors.

With the above details in mind, the insertion and deletion algorithms for line quadtrees are analogous to those of region or point quadtrees. Insertion of a second line segment into a region described by a leaf already containing one line segment causes the leaf to be quartered, the information that was in the original leaf to be distributed among the new leaves, and then the insertion attempt is repeated. The important thing to remember is that when a line segment lies across two leaves, the appropriate intercepts in both leaves must be eight-neighbors of each other. Deletion of line segments is simply a matter of deleting all the information that is specific to that line segment.

6. Conclusions and plans

6.1. Conclusions

This project has developed a set of software tools for use with a quadtree-encoded cartographic database. A query language was developed to make it easier for a user to work with the database. An editing capability was developed to permit database updating. A memory management system was developed for manipulation of maps too large to fit into main memory. Finally, the original database of regions was augmented with a set of point and linear feature data from the same geographical region. Those data were also quadtree encoded, and programs were written to answer queries (points-in-region, lines-meeting-region) that make use of more than one type of data.

6.2. Plans

It is planned to extend the work on this project to evaluate other types of hierarchical representations for regions, linear features, and point data. These representations include

(a) For point data: point quadtrees, K-d trees, GRID files, EXCELL
(b) For linear feature data: Strip trees, edge quadtrees, line quadtrees
(c) For region data: B-tree encoded quadtrees, pyramids, DF-expressions, quadtree Medial Axis Transforms, forest-based methods.

The evaluation will make use of the same data base used on the present project. It will involve timing and storage space studies, and will also investigate the structure's amenability to use in an off-line storage environment.

Other extensions include enhancement of the query language, and an investigation of representations for gray-scale data (in contrast with binary images). All methods will be scrutinized from the viewpoint of their amenability to use in an off-line environment. The evaluation will make use of the same database used in Phases I and II and will consist of studies of time and storage requirements.

Appendix: Facilities used

The computer used during this project was a VAX 11/780 produced by the Digital Equipment Corporation. It has four megabytes of actual memory, six megabytes of virtual memory, a disk fetch speed of approximately 0.6 megabits per second, and a memory cycle speed of approximately 1400 nanoseconds. The wordsize for the VAX is 32 bits broken into four 8-bit bytes. Data is stored on two DD 11/300 disk drives produced by Plessey Peripheral Systems. Each disk drive has a storage capacity of 300 megabytes. The VAX 11/780 runs under the UNIX operating system (Berkley Release 4.1).

The picture output device used by this project is a Grinnell GMR-27 Display Processor. Its memory consists of thirteen 512x512 bitplanes. Twelve of these bitplanes carry color information (four bits for each of the colors: blue, green, and red). The thirteenth bitplane is used for a white overlay capability. The high order eight bitplanes of the twelve color bitplanes can also be displayed to create a grayscale output. The output speed of quadtrees on this device compares favorably to a raster scan output of a picture file, because the GMR-27 can output a rectangle on the display screen directly from the rectangle's coordinates (i.e., a separate command is not necessary for each pixel in the rectangle as is done when a picture file is output in raster scan mode).

Bibliography on quadtrees

[Abel83a] - D.J. Abel and J.L. Smith, A data structure and algorithm based on a linear key for a rectangle retrieval problem, to appear in Computer Vision, Graphics and Image Processing, 1983.

[Abel83b] - D.J. Abel, A B+-tree structure for large quadtrees, to appear in Computer Vision, Graphics and Image Processing, 1983.

[Ahuj83] - N. Ahuja, On approaches to polygonal decomposition for hierarchical image representation, to appear in Computer Vision, Graphics and Image Processing, 1983 (see also Proceedings of the IEEE Conference on Pattern Recognition and Image Processing, Dallas, TX, 1981, 75-80).

[Alex78] - N. Alexandridis and A. Klinger, Picture decomposition, tree data structures, and identifying directional symmetries as node combinations, Computer Graphics and Image Processing 8, 1978, 43-77.

[Alle82] - E. Allen, R. Trigg, and R. Wood, Maryland Artificial Intelligence Group Franz Lisp Environment, Computer Science TR-1226, University of Maryland, College Park, MD, 1982.

[Aoki79] - M. Aoki, Rectangular region coding for image data compression, Pattern Recognition 11, 1979, 297-312.

[Ball81] - D.H. Ballard, Strip trees: A hierarchical representation for curves, Communications of the ACM 24, 1981, 310-321 (see also corrigendum, Communications of the ACM 25, 1982, 213).

[Bent75a] - J.L. Bentley and D.F. Stanat, Analysis of range searches in quad trees, Information Processing Letters 3, 1975, 170-173.

[Bent75b] - J.L. Bentley, Multidimensional binary search trees used for associative searching, Communications of the ACM 18, 1975, 509-517.

[Bent77] - J.L. Bentley, D.F. Stanat, and E.H. Williams Jr., The complexity of fixed radius near neighbor searching, Information Processing Letters 6, 1977, 209-212.

[Bent79a] - J.L. Bentley, Decomposable searching problems, Information Processing Letters 8, 1979, 133-136.

[Bent79b] - J.L. Bentley and J.H. Friedman, Data Structures for range searching, ACM Computing Surveys 11, 1979, 397-409.

[Bent80] - J.L. Bentley and H.A. Maurer, Efficient worst-case data structures for range searching, Acta Informatica 13, 1980, 155-168.

[Bess82] - P.W. Besslich, Quadtree construction of binary images by dyadic array transformations, Proceedings of the IEEE Conference on Pattern Recognition and Image Processing, Las Vegas, NV, 1982, 550-554.

[Blum67] - H. Blum, A transformation for extracting new descriptors of shape, in Models for the Perception of Speech and Visual Form, W. Wathen-Dunn, Ed., M.I.T. Press, Cambridge, MA, 1967, 362-380.

[Burt80] - P.J. Burt, Tree and pyramid structures for coding hexagonally sampled binary images, Computer Graphics and Image Processing 14, 1980, 249-270.

[Burt81] - P. Burt, T.H. Hong, and A. Rosenfeld, Segmentation and estimation of image region properties through cooperative hierarchical computation, IEEE Transactions on Systems, Man, and Cybernetics 11, 1981, 802-809.

[Burt77] - W. Burton, Representation of many-sided polygons and polygonal lines for rapid processing, Communications of the ACM 20, 1977, 166-171.

[Carl82] - W.E. Carlson, An algorithm and data structure for 3D object synthesis using surface patch intersections, Computer Graphics-SIGGRAPH 82 Conference Proceedings 16, 1982, 255-264.

[Come79] - D. Comer, The ubiquitous B-tree, ACM Computing Surveys 11, 1979, 121-137.

[Cook78] - B.G. Cook, The structural and algorithmic basis of a geographic data base, in Proceedings of the First International Advanced Study Symposium on Topological Data Structures for Geographic Information Systems, G. Dutton, Ed., Harvard Papers on Geographic Information Systems, 1978.

[DeCo76] - F. DeCoulon and U. Johnsen, Adaptive block schemes for source coding of black-and-white fac-

simile, Electronics Letters 12, 1976, 61-62 (see also erratum, Electronics Letters 12, 1976, 152).

[DeFl82a] - L. DeFloriani, B. Falcidieno, and C. Pienovi, Triangulated irregular networks in geographical data processing, in Environmental Systems Analysis and Management, S. Rinaldi, Ed., North-Holland, Amsterdam, 1982, 801-811.

[DeFl81b] - L. DeFloriani, B. Falcidieno, G. Nagy, and C. Pienovi, Yet another method for triangulation and contouring for automated cartography. Proceedings of the American Congress on Surveying and Mapping, American Society of Photogrammetry, F.S. Cardwell, R. Black, and B.M. Cole, Eds., Hollywood, FL, 1982, 101-110.

[Dett82] - G. Dettori and B. Falcidieno, An algorithm for selecting main points on a line, Computers and Geosciences 8, 1982, 3-10.

[Duda73] - R.O. Duda and P.E. Hart, Pattern Classification and Scene Analysis, Wiley, New York, 1973.

[Dutt78] - G. Dutton, Ed., Proceedings of the First International Advanced Study Symposium on Topological Data Structures for Geographic Information Systems, Harvard Papers on Geographic Information Systems, 1978.

[Dyer80a] - C.R. Dyer, A. Rosenfeld, and H. Samet, Region representation: boundary codes from quadtrees, Communications of the ACM 23, 1980, 171-179.

[Dyer80b] - C.R. Dyer, Computing the Euler number of an image from its quadtree, Computer Graphics and Image Processing 13, 1980, 270-276.

[Dyer81b] - C.R. Dyer, A VLSI pyramid machine for parallel image processing, Proceedings of the IEEE Conference on Pattern Recognition and Image Processing, Dallas, TX, 1981, 381-386.

[Dyer82] - C.R. Dyer, The space efficiency of quadtrees, Computer Graphics and Image Processing 19, 1982, 335-348.

[East70] - C.M. Eastman, Representations for space planning, Communications of the ACM 13, 1970, 242-250.

[Edel81] - H. Edelsbrunner and H.A. Maurer, A space-optimal solution of general region location, Theoretical Computer Science 16, 1981, 329-336.

[Edel82a] - H. Edelsbrunner, H.A. Maurer, and D.G. Kirkpatrick, Polygonal intersection searching, Information Processing Letters 14, 1982, 74-79.

[Edel82b] - H. Edelsbrunner and H.A. Maurer, On the equivalence of some rectangle intersection problems, Information Processing Letters 14, 1982, 124-127.

[Edel83] - H. Edelsbrunner and J. V. Leeuwen, Multidimensional data structures and algorithms: a bibliography, Institute for Information Processing Report F104, Technical University of Graz, Graz, Austria, 1983.

[Fagi79] - R. Fagin, J. Nievergelt, N. Pippenger, and H.R. Strong, Extendible hashing - a fast access method for dynamic files, ACM transactions on Database Systems 4, 1979, 315-344.

[Fink74] - R.A. Finkel and J.L. Bentley, Quad trees: a data structure for retrieval on composite keys, Acta Informatica 4, 1974, 1-9.

[Fode80] - J. K. Foderaro, The Franz Lisp Manual, The Regents of the University of California, 1980.

[Free74] - H. Freeman, Computer processing of line-drawing images, ACM Computing Surveys 6, 1974, 57-97.

[Frie75] - J.H. Friedman, F. Baskett, and L.J. Shustek, An algorithm for finding nearest neighbors, IEEE Transactions on Computers 24, 1975, 1000-1006.

[Frie77] - J.H. Friedman, J.L. Bentley, and R.A. Finkel, An algorithm for finding best matches in logarithmic expected time, ACM Transactions on Mathematical Software, 1977, 209-226.

[Gibs82] - L. Gibson and D. Lucas, Vectorization of raster images using hierarchical methods, Computer Graphics and Image Processing 20, 1982, 82-89.

[Garg82a] - I. Gargantini, An effective way to represent quadtrees, Communications of the ACM 25, 1982, 905-910.

[Garg82b] - I. Gargantini, Linear octrees for fast processing of three dimensional objects, Computer Graphics and Image Processing 20, 1982, 365-374.

[Gros] - W.I. Grosky and R. Jain, Optimal quadtrees for image segments, IEEE Transactions on Pattern

Analysis and Machine Intelligence 5, 1983, 77-83.

[Hend82] - T.C. Henderson and E. Triendl, Storing feature descriptions as 2-d trees, Proceedings of Pattern Recognition and Image Processing 82, Las Vegas, NV, 1982, 555-556.

[Hoar72] - C.A.R. Hoare, Notes on data structuring, in Structured Programming, O.J. Dahl, E.W. Dijkstra, and C.A.R. Hoare, Eds., Academic Press, London, 1972, 154.

[Horo76] - S.L. Horowitz and T. Pavlidis, Picture segmentation by a tree traversal algorithm, Journal of the ACM 23, 1976, 368-388.

[Huff52] - D.A. Huffman, A method for the construction of minimum-redundancy codes, Proceedings of the IRE 40, 1952, 1098-1101.

[Hunt78] - G.M. Hunter, Efficient computation and data structures for graphics, Ph.D. dissertation, Department of Electrical Engineering and Computer Science, Princeton University, Princeton, NJ, 1978.

[Hunt79a] - G.M. Hunter and K. Steiglitz, Operations on images using quadtrees, IEEE Transactions on Pattern Analysis and Machine Intelligence 1, 1979, 145-153.

[Hunt79b] - G.M. Hunter and K. Steiglitz, Linear transformation of pictures represented by quadtrees, Computer Graphics and Image Processing 10, 1979, 289-296.

[Jack80] - C.L. Jackins and S.L. Tanimoto, Oct-trees and their use in representing three-dimensional objects, Computer Graphics and Image Processing 14, 1980, 249-270.

[Jack82] - C. Jackins and S.L. Tanimoto, Quad-trees, oct-trees, and k-trees - a generalized approach to recursive decomposition of Euclidean space, Department of Computer Science Technical Report 82-02-02, University of Washington, Seattle, WA, 1982.

[Jone81] - L. Jones and S.S. Iyengar, Representation of regions as a forest of quadtrees,, Proceedings of the IEEE Conference on Pattern Recognition and Image Processing, Dallas, TX, 1981, 57-59.

[Kawa80a] - E. Kawaguchi and T. Endo, On a method of binary

picture representation and its application to data compression, IEEE Transactions on Pattern Analysis and Machine Intelligence 2, 1980, 27-35.

[Kawa80b] - E. Kawaguchi, T. Endo, and M. Yokota, DF-expression of binary-valued picture and its relation to other pyramidal representations, Proceedings of the Fifth International Conference on Pattern Recognition, Miami Beach, FL, 1980, 822-827.

[Kawa82] - E. Kawaguchi, T. Endo, and J. Matsunaga, DF-expression viewed from digital picture processing, Department of Information Systems, Kyushu University, Japan, 1982.

[Kede81] - G. Kedem, The Quad-CIF tree: a data structure for hierarchical on-line algorithms, TR 91, Computer Science Department, The University of Rochester, Rochester, NY, 1981.

[Kell71] - M.D. Kelly, Edge detection in pictures by computer using planning, Machine Intelligence 6, 1971, 397-409.

[Kim81a] - C.E. Kim, On the cellular convexity of complexes, IEEE Transactions on Pattern Analysis and Machine Intelligence 3, 1981, 617-625.

[Kim81b] - C.E. Kim and A. Rosenfeld, Digital straightness and convexity, Proceedings of the 13th Annual ACM Symposium on Theory of Computing, Milwaukee, WI, 1981, 80-89.

[Kim82a] - C.E. Kim and A. Rosenfeld, Digital straight lines and convexity of digital regions, IEEE Transactions on Pattern Analysis and Machine Intelligence 4, 1982, 149-153.

[Kim82b] - C.E. Kim, Digital convexity, straightness, and convex polygons, IEEE Transactions on Pattern Analysis and Machine Intelligence 4, 1982, 618-626.

[Kirb79] - R. L. Kirby, R. Smith, P. A. Dondes, S. Ranade, L. Kitchen, and F. Blonder, Interfaces, Subroutines, and Programs for the Grinnell GMR-27 Display Processor on a PDP-11/45 with the UNIX Operating System, Computer Science TR-810, University of Maryland, College Park, MD, 1979.

[Klin71] - A. Klinger, Patterns and search statistics, in Optimizing Methods in Statistics, J.S. Rustagi, Ed., Academic Press, New York, 1971.

[Klin76] - A. Klinger and C.R. Dyer, Experiments in picture representation using regular decomposition, Computer Graphics and Image Processing 5, 1976, 68-105.

[Klin79] - A. Klinger and M.L. Rhodes, Organization and access of image data by areas, IEEE Transactions on Pattern Analysis and Machine Intelligence 1, 1979, 50-60.

[Know80] - K. Knowlton, Progressive transmission of grey-scale and binary pictures by simple, efficient, and lossless encoding schemes, Proceedings of the IEEE 68, 1980, 885-896.

[Knut73] - D.E. Knuth, The Art of Computer Programming, vol. 3, Sorting and Searching, Addison-Wesley, Reading, MA, 1973.

[Knut75] - D.E. Knuth, The Art of Computer Programming, vol. 1, Fundamental Algorithms, Second Edition, Addison-Wesley, Reading, MA, 1975.

[Krau75] - E.F. Krause, Taxicab Geometry, Addison-Wesley, Reading, MA, 1975.

[Lee78] - D.T. Lee and C.K. Wong, Worst-case analysis for region and partial region searches in multidimensional binary search trees and quad trees, Acta Informatica 9, 1978, 23-29.

[Lee80] - D.T. Lee and C.K. Wong, Quintary trees: a file structure for multidimensional database systems, ACM Transactions on Database Systems, 1980, 339-353.

[Lete82] - P. Letelier, personal communication, 1982.

[Li82] - M. Li, W.I. Grosky, and R. Jain, Normalized quadtrees with respect to translations, Computer Graphics and Image Processing 20, 1982, 72-81.

[Lieb78] - H. Lieberman, How to color in a coloring book, Computer Graphics-SIGRAPH 78 Conference Proceedings 12, 1978, 111-116.

[Linn73] - J. Linn, General methods for parallel searching, Technical Report 81, Digital Systems Laboratory, Stanford University, Stanford, CA, 1973.

[Luek78] - G. Lueker, A data structure for orthogonal range queries, Proceedings 19th Annual IEEE Symposium on Foundations of Computer Science, Ann Arbor, MI,

1978, 28-34.

[McCl65] - E.J. McCluskey, Introduction to the Theory of Switching Circuits, McGraw-Hill, New York, 1965, 60-61.

[Mart82] - J.J. Martin, Organization of geographical data with quad trees and least square approximation, Proceedings of the IEEE Conference on Pattern Recognition and Image Processing, Las Vegas, NV, 1982, 458-463.

[Mats83] - T. Matsuyama, L.V. Hao, and M. Nagao, A file organization for geographic information systems based on spatial proximity, to appear in Computer Vision, Graphics and Image Processing, 1983.

[Meag82] - D. Meagher, Geometric modeling using octree encoding, Computer Graphics and Image Processing 19, 1982, 129-147.

[Merr73] - R.D. Merrill, Representations of contours and regions for efficient computer search, Communications of the ACM 16, 1973, 69-82.

[Mins69] - M. Minsky and S. Papert, Perceptrons: An Introduction to Computational Geometry, MIT Press, Cambridge, MA, 1969.

[Mont70] - U. Montanari, On limit properties of digitization schemes, Journal of the ACM 17, 1970, 348-360.

[Mort66] - G.M. Morton, A computer oriented geodetic data base and a new technique in file sequencing, IBM Canada, 1966.

[Nagy79] - G. Nagy and S. Wagle, Geographic data processing, ACM Computing Surveys 11, 1979, 139-181.

[Nash83] - C. Nash and N. Ahuja, Octree representations of moving objects, to appear in Computer Vision, Graphics, and Image Processing, 1983.

[Newm79] - W. Newman and R.F. Sproull, Principles of Interactive Computer Graphics, Second Edition, McGraw-Hill, New York, 1979.

[Niev81] - J. Nievergelt, H. Hinterberger, and K.C. Sevcik, The GRID file: an adaptable, symmetric multi-key file structure, Report 46, Institut für Informatik, ETH, Zurich, Switzerland, 1981.

[Nils69] - N.J. Nilsson, A mobile automaton: an application

of artificial intelligence techniques, Proceedings of the First International Joint Conference on Artificial Intelligence, Washington, DC, 509-520.

[Oliv83] - M.A. Oliver and N.E. Wiseman, Operations on quadtree-encoded images, The Computer Journal 26, 1983, 83-91.

[Omol80] - J.O. Omolayole and A. Klinger, A hierarchical data structure scheme for storing pictures, in Pictorial Information Systems, S.K. Chang and K.S. Fu, Eds., Springer Verlag, Berlin, 1980.

[ORou81a] - J. O'Rourke, Dynamically quantized spaces applied to motion analysis, Technical report JHU-EE 81-1, Electrical Engineering Department, Johns Hopkins University, Baltimore, MD, 1981.

[ORou81b] - J. O'Rourke, Dynamically quantized spaces for focusing the Hough Transform, Proceedings of the Sixth International Joint Conference on Artificial Intelligence, Vancouver, BC, 1981, 737-739.

[Pavl76] - T. Pavlidis, The use of algorithms of piecewise approximations for picture processing applications, ACM Transactions on Mathematical Software 2, 1976, 305-321.

[Pavl79] - T. Pavlidis, Filling algorithms for raster graphics, Computer Graphics and Image Processing 10, 1979, 126-141.

[Peuc76] - T. Peucker, A theory of the cartographic line, International Yearbook of Cartography, 1976, 134-142.

[Peuq79] - D.J. Peuquet, Raster processing: an alternative approach to automated cartographic data handling, American Cartographer 6, 1979, 129-139.

[Piet82] - M. Pietikainen, A. Rosenfeld, and I. Walter, Split-and-link algorithms for image segmentation, Pattern Recognition 15, 1982, 287-298.

[Pfal67] - J.L. Pfaltz and A. Rosenfeld, Computer representation of planar regions by their skeletons, Communications of the ACM 10, 1967, 119-122.

[Rana80] - S. Ranade, A. Rosenfeld, and J.M.S. Prewitt, Use of quadtrees for image segmentation, Computer Science TR-878, University of Maryland, College Park, MD, 1980.

[Rana81a] - S. Ranade, Use of quadtrees for edge enhancement, IEEE Transactions on Systems, Man, and Cybernetics 11, 1981, 370-373.

[Rana81b] - S. Ranade and M. Shneier, Using quadtrees to smooth images, IEEE Transactions on Systems, Man, and Cybernetics 11, 1981, 373-376.

[Rana82] - S. Ranade, A. Rosenfeld, and H. Samet, Shape approximation using quadtrees, Pattern Recognition 15, 1982, 31-40.

[Redd78] - D.R. Reddy and S. Rubin, Representation of three-dimensional objects, CMU-CS-78-113, Computer Science Department, Carnegie-Mellon University, Pittsburgh, PA, 1978.

[Rise77] - E.M. Riseman and M.A. Arbib, Computational techniques in the visual segmentation of static scenes, Computer Graphics and Image Processing 6, 1977, 221-276.

[Rose66] - A. Rosenfeld and J.L. Pfaltz, Sequential operations in digital image processing, Journal of the ACM 13, 1966, 471-494.

[Rose74] - A. Rosenfeld, Digital straight line segments, IEEE Transactions on Computers 23, 1974, 1264-1269.

[Rose76] - A. Rosenfeld and A.C. Kak, Digital Picture Processing, Academic Press, New York, 1976.

[Rose79] - A. Rosenfeld, Digital Topology, American Mathematical Monthly 86, 1979, 621-630.

[Rose82a] - A. Rosenfeld, H. Samet, C. Shaffer, and R.E. Webber, Application of hierarchical data structures to geographical information systems, Computer Science TR-1197, University of Maryland, College Park, MD, 1982.

[Rose82b] - A. Rosenfeld and C.E. Kim, How a digital computer can tell whether a line is straight, American Mathematical Monthly 89, 1982, 230-235.

[Rose83a] - Rosenfeld, A., Picture processing: 1982, Computer Graphics and Image Processing 22, 1983.

[Rose83b] - A. Rosenfeld, Ed., Multiresolution Image Processing and Analysis, Springer Verlag, Berlin, 1983.

[Ruto68] - D. Rutovitz, Data structures for operations on digital images, in Pictorial Pattern Recognition, G.C. Cheng et al., Eds., Thompson Book Co., Washington, DC, 1968, 105-133.

[Same80a] - H. Samet, Region representation: quadtrees from boundary codes, Communications of the ACM 23, 1980, 163-170.

[Same80b] - H. Samet, Region representation: quadtrees from binary arrays, Computer Graphics and Image Processing 18, 1980, 88-93.

[Same80c] - H. Samet and A. Rosenfeld, Quadtree structures for image processing, Proceedings of the Fifth International Conference on Pattern Recognition , Miami Beach, FL, 1980, 815-818.

[Same80d] - H. Samet, Deletion in two-dimensional quad trees, Communications of the ACM 23, 1980, 703-710.

[Same81a] - H. Samet, An algorithm for converting rasters to quadtrees, IEEE Transactions on Pattern Analysis and Machine Intelligence 3, 1981, 93-95.

[Same81b] - H. Samet, Connected component labeling using quadtrees, Journal of the ACM 28, 1981, 487-501.

[Same81c] - H. Samet, Computing perimeters of images represented by quadtrees, IEEE Transactions on Pattern Analysis and Machine Intelligence 3, 1981, 683-687.

[Same82a] - H. Samet Neighbor finding techniques for images represented by quadtrees, Computer Graphics and Image Processing 18, 1982, 37-57.

[Same82b] - H. Samet, Distance transform for images represented by quadtrees, IEEE Transactions on Pattern Analysis and Machine Intelligence 4, 1982, 298-303.

[Same82c] - H. Samet and R.E. Webber, Line quadtrees: a hierarchical data structure for encoding boundaries, Proceedings of the IEEE Conference on Pattern Recognition and Image Processing, Las Vegas, NV, 1982, 90-92.

[Same82d] - H. Samet, Data structures for quadtree approximation and compression, Computer Science TR-1209, University of Maryland, College Park, MD, 1982.

[Same82e] - H. Samet, Reconstruction of quadtrees from quadtree medial axis transforms, Computer Science TR-1224, University of Maryland, College Park, MD, 1982.

[Same82f] - H. Samet, A top-down quadtree traversal algorithm, Computer Science TR-1237, University of Maryland, College Park, MD, 1982.

[Same83a] - H. Samet, A quadtree medial axis transform, to appear in Communications of the ACM, 1983 (also University of Maryland Computer Science TR-803).

[Same83b] - H. Samet, Algorithms for the conversion of quadtrees to rasters, to appear in Computer Vision, Graphics and Image Processing, 1983 (also University of Maryland Computer Science TR-979).

[Same83c] - H. Samet and E. V. Krishnamurthy, A quadtree-based matrix manipulation system, in progress.

[Same83d] - H. Samet and R. E. Webber, Using quadtrees to represent polygonal maps, Proceedings of Computer Vision and Pattern Recognition 83, Washington, DC, 1983, 127 - 132.

[Sham75] - M.I. Shamos and D. Hoey, Closest-point problems, Proceedings of the Sixteenth Annual IEEE Symposium on the Foundations of Computer Science, Berkeley, CA, 1975, 151-162.

[Shan80] - U. Shani, Filling regions in binary raster images: a graph-theoretic approach, Computer Graphics-SIGGRAPH 80 Conference Proceedings 14, 1980, 321-327.

[Shne81a] - M. Shneier, Calculations of geometric properties using quadtrees, Computer Graphics and Image Processing 16, 1981, 296-302.

[Shne81b] - M. Shneier, Path-length distances for quadtrees, Information Sciences 23, 1981, 49-67.

[Shne81c] - M. Shneier, Tw hierarchical linear feature representations: edge pyramids and edge quadtrees, Computer Graphics and Image Processing 17, 1981, 211-224.

[Sloa79] - K.R. Sloan Jr. and S.L. Tanimoto, Progressive refinement of raster images, IEEE Transactions on Computers 28, 1979, 871-874.

[Sloa81] - K.R. Sloan Jr., Dynamically quantized pyramids,

Proceedings of the Sixth International Joint Conference on Artificial Intelligence, Vancouver, BC, 1981, 734-736.

[Smit79] - A.R. Smith, Computer Graphics-SIGGRAPH 79 Conference Proceedings 13, 1979, 276-283.

[Suth74] - I.E. Sutherland, R.F. Sproull, and R.A. Schumacker, A characterization of ten hidden-surface algorithms, ACM Computing Surveys 6, 1974, 1-55.

[Tamm81] - M. Tamminen, The EXCELL method for efficient geometric access to data, Acta Polytechnica Scandinavica, Mathematics and Computer Science Series No. 34, Helsinki, 1981.

[Tamm83] - M. Tamminen, Encoding pixel trees, Laboratory of Information Processing Science, Helsinki University of Technology, Espoo, Finland, 1983.

[Tani75] - S. Tanimoto and T. Pavlidis, A hierarchical data structure for picture processing, Computer Graphics and Image Processing 4, 1975, 104-119.

[Tani76] - S. Tanimoto, Pictorial feature distortion in a pyramid, Computer Graphics and Image Processing 5, 1976, 333-352.

[Tani80] - S. Tanimoto and A. Klinger, Eds., Structured Computer Vision, Academic Press, New York, 1980.

[Tous80] - G.T. Toussaint, Pattern recognition and geometrical complexity, Proceedings of the Fifth International Conference on Pattern Recognition, Miami Beach, FL, 1980, 1324-1346.

[Trop81] - H. Tropf and H. Herzog, Multidimensional range search in dynamically balanced trees, Angewandte Informatik 2, 1981, 71-77.

[Uhr72] - L. Uhr, Layered "recognition cone" networks that preprocess, classify, and describe, IEEE Transactions on Computers 21, 1972, 758-768.

[Warn69] - J.L. Warnock, A hidden surface algorithm for computer generated half tone pictures, Computer Science Department TR 4-15, University of Utah, Salt Lake City, UT, 1969.

[Webe78] - W. Weber, Three types of map data structures, their ANDs and NOTs, and a possible OR, in Proceedings of the First International Advanced Study Symposium on Topological Data Structures for

Geographic Information Systems, G. Dutton, Ed., Harvard Papers on Geographic Information Systems, 1978.

[Will78] - D.E. Willard, Predicate-oriented database search algorithms, Report TR-20-78, Harvard University Aiken Laboratory, 1978.

[Will82] - D.E. Willard, Polygon retrieval, SIAM Journal on Computing 11, 1982, 149-165.

[Wood82] - J.R. Woodwark, The explicit quadtree as a structure for computer graphics, The Computer Journal 25, 1982, 235-238.

[Wu82] - A.Y. Wu, T.H. Hong, and A. Rosenfeld, Threshold selection using quadtrees, IEEE Transactions on Pattern Analysis and Machine Intelligence 4, 1982, 90-94.

[Yau81] - M. Yau and S.N. Srihari, Recursive generation of hierarchical data structures for multidimensional digital images, Proceedings of the IEEE Conference on Pattern Recognition and Image Processing, Dallas, TX, 1981, 42-44.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER ETL-0337	2. GOVT ACCESSION NO. AD-A134 999	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) APPLICATION OF HIERARCHICAL DATA STRUCTURES TO GEOGRAPHICAL INFORMATION SYSTEMS (PHASE II)		5. TYPE OF REPORT & PERIOD COVERED Contract Report 7/29/82 - 6/29/83
		6. PERFORMING ORG. REPORT NUMBER TR-1327
7. AUTHOR(s) Hanan Samet Azriel Rosenfeld		8. CONTRACT OR GRANT NUMBER(s) DAAK70-81-C-0059
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer Vision Laboratory University of Maryland College Park, MD 20742		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS R3205HT09
11. CONTROLLING OFFICE NAME AND ADDRESS U.S. Army Engineer Topographic Labs. Fort Belvoir, VA 22060		12. REPORT DATE September 30, 1983
		13. NUMBER OF PAGES 81
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Query language
Quadtree editor
Storage management
Geographical information systems
Data structures
Quadtrees

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This document is the final report for Phase II of an investigation of the application of hierarchical data structures to geographical information systems. It describes a set of software tools developed for use with a quadtree encoded database containing area, point, and line data. Included in this software is an English-like query language, an editing capability to permit database updating, and a memory management system to allow manipulation of maps too large to fit into main memory.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

END

FILMED

1-84

DTIC